EXPLORING SOFTWARE SECURITY APPROACHES AND THEIR LIMITATIONS IN SOFTWARE DEVELOPMENT LIFECYCLE

BY

NABIL MOHAMMED ABDO MOHAMMED

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

MAY, 2015.

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by NABIL MOHAMMED ABDO MOHAMMED under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Dr. Mahmoud Niazi
(Advisor)

Dr. Abdulaziz Alkhoraidly
Department Chairman

Dr. Mohammad Alshayeb
(Member)

Dr. Salam A. Zummo
Dean of Graduate Studies

Dr. Sajjad Mahmood
(Member)

7/16/15

Date

Dedicated To

My parents

ACKNOWLEDGMENTS

First and foremost thanks to Allah (SWT) for giving me the strength, patience and ability to accomplish my thesis work. Peace and blessing of Allah be upon his last messenger Prophet Mohammed (Sallallahu-Alaihe-Wasallam), who guided us to the right path.

I would like to express my thankfulness to my thesis Advisor Dr.Mahmood Niazi for the continuous support and encouragement rendered toward me. I would also like to thank my committee members Dr. Mohammad Alshayeb and Dr. Sajjad Mahmood, for their guidance throughout my thesis work.

I am very grateful for the love and unlimited support of my parents and my family who carved the path to my successful journey in the field of research and education. Finally, I would like to express my deepest thanks to all of my friends who helped me to accomplish this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SDLC	:	Software Development Lifecycle
XP	:	Extreme Programming
CC	:	Common Criteria
SLR	:	Systematic Literature Review
CERT	:	Computer Emergency Response Team
BOF	:	Buffer OverFlow
XSS	:	Cross Site Scripting
SQL	:	Structural Query Language
SQLIA	:	SQL Injection Attack
UML	:	Unified Modelling Language
OWASP	:	Open Web Application Security Project
RBAC	:	Role Based Access Control
MDS	:	Model Driven Security
CLASP	:	Comprehensive Lightweight Application Security Process
SSAI	:	Software Security Assessment Instrument

ABSTRACT

Full Name : [NABIL MOHAMMED ABDO MOHAMMED]

Thesis Title : [Exploring software security approaches and their limitations in software development lifecycle]

Major Field : [Information and Computer Science]

Date of Degree : May ,2015

Software security is only considered in the later stages of software development with the incorporation of security concerns as an afterthought. As a consequence, the risk of introducing new security vulnerabilities into various stages of software development lifecycles increases. Research evidence has proven that approaches to address security-related concerns are insufficient and could likely cause costly reworks in addition to all the intangible consequences caused by a security breach. To avoid these costly mistakes, security concerns need to be addressed from the beginning of software development lifecycles all the way through to deployment and maintenance. Several approaches have been proposed in the literature for incorporating security into the SDLC from the requirements gathering phase until the maintenance and deployment, along with recommended tools to support a security-centric software development lifecycle. Despite the importance of these approaches, little research has been carried out to investigate these approaches and their limitations in a systematic manner. In this thesis, we propose to explore and identify software security approaches and their limitations in the software development lifecycle. Systematic Literature Review (SLR) and Snowballing are the research methodology used to guide us in finding the answer of our research questions. In total, we selected and categorized 165 articles. Several software security approaches have

been identified that provided security checks in various software development phases, such as requirements, design, and coding. Also, the results show that the most frequently cited approaches are static analysis and dynamic analysis that provide security checks in the coding phase. Furthermore, this study shows that the significant number of studies in this review considered security checks around the coding stage of software development. Finally, the limitations of existing software security approaches of incorporating security check in to software development- whether for identified existing software security approaches or general challenges and limitations - are identified. This work assists software development organizations in better understanding the existing software security approaches used in the software development lifecycle and their limitations. It can also provide other researchers with a firm basis on which to develop new software security approaches and address any of the identified limitations. We hope that our research will facilitate any future research on enhance the identified software security approaches and address their limitations.

ملخص الرسالة

الاسم الكامل: نبيل محمد عبده محمد

عنوان الرسالة: استكشاف أساليب أمن البرمجيات وتحدياتها في دروة تطوير البرمجيات (SDLC)

الدرجة العلمية: ماجستير العلوم

التخصص: علوم الحاسب الألي

تقليديا, يُهتم بأمن البرمجيات فقط في المراحل المتأخرة من تطوير البرمجيات مع دمج نشاطات وممارسات الأمن في المراحل اللاحقة . ونتيجة لذالك, فإن مخاطر إدخال الثغرات الأمنية الجديدة في مختلف مراحل تطوير البرمجيات في ازدياد. الادلة البحثية أثبتت أن الاساليب لمعالجة أمن البرمجيات في المراحل اللاحقه في دورة تطوير البرمجيات (Software development lifecycle) غير فعالة ويمكن على الارجح ان تسبب في اعادة صياغه أو مراجعة مكلفة ,بالاضافة الى عواقب غير ملموسة الناجمه عن الخرق الامني. لتجنب هذه الاخطاء المكلفة والثغرات الامنية , المخـاوف الأمنية (security concerns) ينبغى معالجتها من بداية تطوير البرمجيات ابتداء من مرحلة جمع المتطلبات والتصميم وصولا الى كتابة الأكواد والصيانة والنشر. هناك العديد من الاساليب تم اقتراحها في البحوث السابقة لدمج أمن البرمجيات الى دورة تطوير البرمجيات ابتداء من مرحلة جمع المتطلبات وصولا الى الصيانة والنشر , جنبا الى جنب مع الادوات الموصى بها لدعم دورة تطوير البرمجيات. على الرغم من اهمية هذه الاساليب والمناهج , قليل من البحث تم تنفيذه لتحديد واستكشاف هذه المناهج وتحدياتها بطريقة ممنهجة. في هذه الاطروحة , نقوم باستكشاف وتحديد اساليب ومناهج أمن البرمجيات وتحدياتها في دورة حياة تطوير البرمجيات. طريقة البحث المستخدمة هي المراجعه الممنهجه للادب (SLR) و السنوبولنج (Snowballing), والذين يقومان بتوجيهينا في العثور على إجابة سؤال البحث لدينا .في المجموع, تم اختيار وتصنيف ١٦٥ مقالا. العديد من مناهج واساليب أمن البرمجيات التي تقدم الفحص والتحقيق الامني في دور حياة تطوير البرمجيات تم ايجادها واستكشافها ابتداء من مرحلة جمع المتطلبات مرورا بتصميمها وكتابة الاكواد. بالاضافه الى ذالك , نتائج هذه الدراسه أظهرت ان التحليل الثابت (static analysis) والتحليل الديناميكي(Dynamic analysis) اكثر ذكرا ودراسة في البحوث المنشورة في هذا الموضوع. ومن ناحية اخرى اظهرت هذه الدراسة , ان عدد كبير من الدراسات اهتمت بتقديم الفحص الامني خلال مرحلة كتابة الاكواد في دورة حياة تطوير البرمجيات. بالاضافة الى ذالك ,تم تحديد تحديات وعوائق مناهج واساليب تطوير أمن البرمجيات –سواء في الاساليب الموجوده او العوائق العامة لتطوير برمجيات أمنه - في دورة حياة تطوير البرمجيات. هذا العمل يساعد منظمات تطوير البرمجيات في فهم مناهج واساليب تطوير أمن البرمجيات خلال دورة حياة تطوير البرمجيات وتحدياتها. وكذالك هذه الدراسة تزود الباحثين والممارسين اساسا متينا لتطوير اساليب ومناهج جديدة لدعم تطوير برمجيات امنه وكذالك اقتراح حلول للعوائق والتحديات في الاساليب والمناهج المنشورة مسبقا. نأمل أن هذه الرسالة ستسهل اي عمل في المستقبل في تحسين مناهج واساليب تطوير أمن البرمجيات وحل عوائقها وتحدياتها.

CHAPTER 1

INTRODUCTION

1.1 Overview

Software-intensive systems have become an inseparable part of our lives today. Our dependence on software systems is very high in several areas of our daily activities, such as telecommunications, financial services, electronics, home appliances, transportation, and more. As the software system is involved in various aspects of society, security becomes an important issue and a vital requirement for the software system. Many security issues such as confidentiality, availability and integrity need to be preserved in order to consider software as secure [1].

Traditionally, software security is considered only in the later stages of software development, by incorporating security concerns as an afterthought. As a consequence, the risk of introducing new security vulnerabilities into various stages of software development lifecycles will be increased. Following the traditional method of securing Software has led to the Penetrate and Patch approach, in which the security specialist tries to assess the software by breaking it from its environment via exploiting common security vulnerabilities. Successful penetration leads to patch development and deployment of the identified vulnerabilities. Security has been always treated as an add-on feature in the software development lifecycle, and is addressed by security

professionals using firewalls, proxies, intrusion prevention systems, antivirus and platform security. Software is at the root of all common computer security problems, hence the reason why hackers don't create security holes, but rather exploit them. Security holes in software applications are the result of bad design and poor implementation of software systems and applications. Unfortunately, cryptographic components as well as other defensive mechanisms, such as intrusion detection systems and firewalls, which are supplemented to a software system towards the end of the development cycle, are insufficient and may lead to costly reworks [2]. Research evidence has proven that such approaches to address security-related concerns are insufficient and will likely cause costly reworks in addition to any intangible consequences caused by a security breach. To avoid these costly reworks, security concerns need to be addressed from the beginning of software development lifecycles (i.e. from the requirements gathering until deployment and maintenance). To this end, secure software engineering has recently become a very active area of research. Like all other engineering disciplines, software engineering involves a structured sequence of stages to develop a software product. These stages are known as the software development lifecycle. The main stages, whether using traditional or agile methodologies, are: requirements analysis, design, implementation, testing, deployment and maintenance. However, none of the traditional methodologies used for software development lifecycles have considered security as a deliverable in any of the stages of the lifecycle. Security has been always treated as an add-on feature in software, which explains the reason behind security bugs and flaws that are exploited by hackers today.

Therefore, the goal of secure software engineering is to target the software security vulnerabilities by considering security concerns and development approaches from the beginning of the software development lifecycles (i.e. from the requirements gathering until the end of the process). Secure software engineering is the procedure of achieving security purposes through build, design and test the software. Also, Software security is different from application security in that application security is about protecting software after development and deployment. It usually includes various protection mechanisms such as firewalls, antivirus and intrusion detection systems [3][4]. Several approaches have been proposed in the literature that is used for incorporating security into the SDLC from the requirements gathering until maintenance and deployment, along with tools to support a security-centric software development lifecycle. Despite the importance of these approaches, little research has been carried out to investigate these approaches and their limitations in a systematic manner.

During the last few years, a number of papers have focused on secure software development, some of which have carried out reviews and comparison studies on the issue. However, most of these reviews focused only on the secure software engineering at the requirements engineering phase of the SDLC and others concentrated only on special software development methodologies, such as Agile or XP. After performing preliminary searches aimed at both identifying existing systematic reviews and assessing the volume of potentially relevant studies, we can highlight a few works in a summary of a small group of security approaches in the security requirement of the requirements phase, such as [5]–[10], as well as a review of security approaches for specific domains such as web application[11]. There were also some reviews that investigated software security in

specific development methodologies such as agile and XP [12] [13]. However, none of them performed a review focused on software security approaches that cover all stages of software development lifecycles in a systematic manner and their limitations, and none of those reviews documented the systematic processes for selecting the initiatives (primary studies). Thus, there is a need to investigate the available security approaches and their limitations—as well as the stages in which these approaches are incorporated—in a systematic manner, to identify the gap in this area for further contribution by both the researcher and the practitioner.

With this focus, the objective of this research is to identify an available approaches for secure software development in a systematic manner, through commonly used methodology in software engineering called a systematic literature review (SLR). An ultimate outcome of this research is to aid software development organizations with a sound knowledge of the existing secure software development approaches, as well as the stage in which these approaches are incorporated. Also, we will assist the software development organization in better understanding the limitations of existing software security approaches used in the software development lifecycle and to provide other researchers with a firm basis on which to develop new software security approaches.

1.2 Thesis Objectives

The overarching objective of this research is to identify the existing software security approaches used in the software development lifecycle as well as its limitations, and to provide other researchers with a firm basis on which to develop new software security approaches.

The objective of this research is two-fold:

1. Identify different software security approaches used in the software development lifecycle.

RQ1: What approaches are available for secure software development?

RQ2: At which stage of the software development lifecycle is the software security approach incorporated?

RQ3: Which researchers are most active in software security?

RQ4: What are the main venues for publications on software security?

2. Identify the existing limitations of software security approaches used in SDLC.

RQ5: What are the limitations of software security approaches used in the software development lifecycle?

A systematic approach will be employed with the intention of achieving the thesis objectives to identify the software security approaches for building security from the beginning of the software development lifecycles, as well as the possible existing limitations of these security approaches. This approach will be implemented by using the concepts of "systematic literature review" (SLR). Additionally, snowballing from the list of references of the identified articles used is another method used in this research (i.e., to identify additional relevant articles through the references lists of the articles found using the search strings, as well as the articles identified through manual search using Google Scholar to answer the (RQ5). Both backward snowballing from the lists of references and forward snowballing which is finding the citations to the papers, were included in this research.

The contribution of the thesis assists software development organizations in better understanding the existing software security approaches used in the software development lifecycle and its limitations, as well as to provide other researchers and practitioners with a firm basis, solid foundation and a body of knowledge on which to develop new software security approaches.

The following approaches are used as a guide for answering our research questions:

1. Identify different software security approaches used in the software development lifecycle by using the systematic literature review methodology.
2. Identify, through the use of snowballing, the existing limitations of software security approaches used in the software development lifecycle.
3. Analyze the results of step 1 and 2 to provide a comprehensive knowledge and to achieve our objectives.

1.3 Research Methodology

In order to achieve our objectives, we have designed an appropriate research methodology in which data will be collected from the published literature (i.e., a systematic literature review process and snowballing). These two processes will give us confidence in the reliability of the data collected. A systematic literature review is a defined and methodical way to summarize the empirical evidence concerning a treatment or technology, in order to identify missing areas in current research, or to provide background in order to justify new research. A systematic literature review requires considerably more effort than conventional literature reviews, but provides a much stronger basis for making claims about research questions [14]. Hence, the SLR was an

appropriate research method for our research, which is aimed at identifying the software security approaches for building security from the beginning of SDLC. We will follow the SLR guidelines proposed by Kitchenham and Charters [14] for performing the SLR, which contains three main processes identified:

1) ***Planning the review:*** By specifying the research questions and developing the review protocol which contains the search strategy, and by identifying search strings derived from the research questions, scopes and methods. Additionally, the quality assessment of selected studies as well as the inclusion and exclusion criteria and data extraction forms will be used.

2) ***Conducting the review:*** By identifying relevant researches and selecting the primary studies from them, we will then assess the study quality, extracting the required data. Finally, we will synthesize the extracted data, checking the most frequent approaches that are used for incorporating security concerns into the SDLC and the most frequent phases where the security approaches are emphasized. This categorization will help in identifying the most neglected stage in terms of security so that new room can be opened for further research. Additionally, we will analyze studies based on the countries where they were conducted, the active researchers and on the publication venue of the primary studies that contribute to the topic.

3) ***Reporting the review*****:** We will write up the final report.

After the results (i.e., software security approach for building security into software development lifecycles) have been identified by SLR, we will identify the possible limitations and challenges of incorporating security into the SDLC using snowballing from the list of references of the identified articles via the SLR and the articles identified

through manual search using Google Scholar. Both results will build a comprehensive mapping study about building security from the ground up through integrating security into the SDLC.

Our research methodology and approach can hence be summarized into the following phases:

Phase 1: Systematic Literature Review (SLR)

In the first phase we will start the systematic literature review. We have identified the primary resources and research database as follows: ACM Digital Library, ScienceDirect, IEEEXplore, SprinerLink and John Wiley Online Library.

Phase 2: Snowballing

In this phase, we will identify the limitations and challenges of software security approaches for building security into the SDLC using snowballing from the list of references of the identified articles via SLR and the articles identified through manual search using Google Scholar.

Phase 3: Interpretations and Analysis

The results compiled from the SLR and snowballing will be interpreted and analyzed in alignment with research objectives in order to answer the research questions.

Phase 4: Conclusion

The conclusion of the entire effort of this research will be presented.

Phase 5: Thesis Writing

Complete the thesis write-up.

1.4 Thesis Outline

The remaining sections of the thesis are organized as follows. Chapter 2 presents basic terminology and background information on software security and secure development. Chapter 3 presents the state-of-the-art literature review in the field pointing out the gaps in the literature which is addressed by this thesis. The literature review will compose a body of knowledge necessary to justify our purpose of the research. Chapter 4 addresses the research methodology of our research. Results are illustrated in tabulated and charted format in chapter 5. It will be accompanied by extensive interpretation and analysis in alignment with the research objectives. Chapter 6 draws a conclusion on our research.

CHAPTER 2

BACKGROUND AND OVERVIEW

This chapter presents basic terminology and background information on software security and software development lifecycles. Section 2.1 explains software security concepts, while section 2.2 presents the software development lifecycles, concepts and activities, and section 2.3 and section 2.3 discuss security into the SDLC and the important of a systematic literature review.

2.1 Software Security

I. Software security concepts:

There are some definitions of software security that has been found in the literature as follows:

-*"Software Security is the ability of the software to resist, tolerate, and recover from events that intentionally threaten its dependability." [3]*

-*"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." [15]*

-*"The idea of engineering software that continues to function correctly under malicious attack." [16]*

-*"The process of designing, building, and testing software for security." [15]*

-*"Defends against software exploits by building software to be secure in the first place, mostly by getting the design right (which is hard) and avoiding common mistakes ." [17]*

-*"Software Security is a system-wide issue that takes into account both security mechanisms (such as access control) and design for security (such as robust design that make software attacks difficult)." [15]*

Software security has been defined by different people, and no standard definition has been agreed upon. We can observe that most of the above definitions are concerned about building secure software which is actually means to design and implement secure software from the beginning of software development.

II. Software security terminology

- **Asset** is anything valuable, and needs to be protected. It is the "target of threats, the possessors of exposures, or the beneficiary of countermeasures". According to McGraw and his colleagues [18], several things could be considered as an asset, such as components or complete systems, information or data stored by software, code—whether binary or source, and services supplied by the software. Also, severe consequences such as physical injury, financial loss, and sometimes even death due to the effective exploitation any of these assets [18].

- **Software vulnerability** is "a weakness in the security system, for example, in procedures, design, or implementation that might be exploited to cause loss or harm" [19]. The errors in the software may make it vulnerable, and these errors can be found in different stages such as requirement specification, design, or coding of a system [19]. Software vulnerabilities are classified into two main categories [3] :

- ✓ Design-level: at this level, the vulnerability may occur as a result of a design flaw.
- ✓ Implementation-level: at this level, the vulnerability may occur as a result of a bug in the code. In this type, the attackers can exploit the vulnerability easily to achieve their purposes.

Defect is a latent problem that lays for years [20].

Bug Problem that exists in the code during the implementation stage [3].

- **Software Security Error**

In the literature, the researchers use the term *vulnerability* instead of *error* and most of the authors do not differentiate between them [21]. It has been defined as "a tangible manifestation of a mistake in any of the SDLC artifacts (requirement specifications, design, or source code) of a piece of software that leads to a vulnerability" [21][22].

A software security error is categorized into three types of errors:

a) Requirement error: this error can happen due to an incorrect or missing requirement as a result of a mistake made by the requirement engineer who is responsible for specifying the requirements.

b) Design error - improper logical decision (whether in the representation of the decision or in the decision itself) in the design phase of the software development lifecycles.

c) Coding error - mistake made by the coder (implementer) in the implementation stage of software development lifecycles that leads to represent the design decision incorrectly in the source code.

- **Software Security Requirements**

Software security requirements are the requirements that are needed in order to mitigate software security errors in the software development lifecycle [22]. More precisely, a software security requirement can be considered as a constraint or control which will mitigate the chance of vulnerability if it is implemented in a suitable manner. More attention should be given to software security requirements that are specified for any SDLC artifact because a software security error can exist at any stage of the software development lifecycles artifact [22].

- **Risk**

The risk is the product of the probability of the occurrence of the attack multiplied by the damage of that successful attack on different assets of the software [18][22].

- **Risk Specification**

Risk specification or risk analysis is calculating the risk. This process could be done in any stage of SDLC. Risk analysis concerns existing vulnerabilities, attacks and their impacts, and the likelihood of future attacks [22][18]. Also, threat modeling [19] is an approach for identifying the possibility of the threats to a piece of software. This approach considers the software assets, occurrence of the attack and attackers' goals.

- **Attack Surfaces**

Possible access points that help the attacker (whether an entity or a person) with potential interaction with the software and intentionally attempting to attack it, such as user interfaces, data files and configuration files. The more entry points, the more attack surfaces and vice versa.

III. Software security and information security

Nowadays, software is the critical issue in computer security since software holes are prevalent, and the problem is still growing. Moreover, the problem may become much worse in the future due to the fact that [23]:

- New software operates in networked environments which is vulnerable to many hostile attacks.
- New systems that are extended by Java VMs and .Net runtime environments become more popular, which lead to mobile code risk.
- The number of complicated and complex types of software is growing.

The fundamental way of solving computer security problems is by making software secure. However, the question here is, "What is the most effective way to protect software from vulnerability?" To answer this question, the difference between a software security and an application security are clarified. The software security is about design and implement secure software through building secure software from the beginning of SDLC. This concept addresses critical issues such as software security requirements, designs for security, security flaws and security tests. It is mostly concerned with designing and implementing software to be secure, as well as training software developers, designers, and users about securing the software through the design, implementation and testing of software before deployment [2] [10].

Application security is about protecting the software after development. This concept addresses critical issues such as protecting against malicious code, input validation of the program, and making software use certain policies with technological

solutions. Application security is mainly about looking for security problems and fixing them after the attacker exploits them. However, this approach concerns security symptoms in a reactive way, ignoring the original cause of the problem.

Both concepts are related to the idea of the prevention of software exploitation. Software security mitigates the chance of exploitation through building the secure software in the early stages of development, mostly by incorporating security into the SDLC. On the other hand, application security mitigates the chance of exploitation by enforcing reasonable policy about what kinds of things can run, how they can change and what the software does as it runs [2] [10].

In order to develop better software, building the software to be secure in the first place by solving problems that are found in the design and implementation stage of software, is better than finding and fixing security problems after the software is built [23]. This will reduce the overall development cost of a product and this is what we are care about in this research.

IV. How is security addressed?

A number of approaches have evolved to address software security. Following are three major approaches used to address security in software [24].

- Penetrate and patch
- Secure operational environment
- Secure software engineering

In the *penetrate and patch* approach, a software product is released to the public after completion. Any vulnerability found is fixed by applying patches. Although it is the

most common approach, to apply patches after finding vulnerabilities is a hundred times [24]–[26] more expensive than if the issues were fixed during development. Most of the time, more vulnerabilities are introduced while applying patches [11] [13] [14]. Securing the operational environment [24] relies on the external devices to the software systems such as firewalls and protection mechanisms. It can provide external security to the software, but helps very little against design and implementation attacks. Moreover, operational environment security is only possible after launching the operational product [24]. The idea behind secure software engineering is to implement well-structured processes and mechanisms from the early phases of software development (i.e., requirement elicitation) [24]. Secure software engineering starts from the requirements phase and is reflected in the entire stage of the SDLC [28]–[31].

i. The need for software security:

Software security has not been given the appropriate attention in recent years. This does not mean that it has not been discussed before, but it would seem that there are some misunderstandings in the concept and the way it should be practiced. A group of researchers and authors began publishing security books in 1999, and started discussing the best way to incorporating security into systems [32]. Incorporating security is often considered as an add-on feature to a system when the development lifecycle is completed. In many organizations, the responsibility of security is left to a few infrastructure people who set up the intrusion detection (IDS), antivirus and firewalls [15]. These people are not developers, architects or designers.

The concept of software security has been neglected by requirement analysts, software implementers, and design architects. They have given it little or no attention during the development process, which leads to dire consequences as security problems show up in the software. CERT (CERT/CC) Coordination Centre has reported about 90% of the security problems due to the exploitation in the development and design flaws [32] and more of them due to bad style in the coding, such as BOF and XSS. Also, these flaws have been exploited even with the existence of the security afterthought approaches such as firewalls, intrusion detectors and antivirus programs.

In addition to the factors mentioned above, many other factors need to be considered that support the need for software security, such as connectivity, complexity and extensibility [33]. Software is getting bigger and bigger in size due to having to deal with huge tasks. This has led to a lot of flaws at the design and coding levels. Also, extensibility can help in providing a cheaper way to update the software such as in Microsoft's .Net and Sun Microsystems' Java that accept code and update. This feature can lead to making the software vulnerable to malicious code. Furthermore, during the last decade, there has been a huge evolution related to connectivity. The growth of local area networks and wireless area networks, as well as the internet, has modernized the connectivity in the world of computers. This helps the attacker to hack systems through remote access from the networks. All of these factors help attackers to achieve their purposes and make software easier to exploit.

Due to ineffective techniques such as antivirus programs, intrusion detectors and firewalls, neglecting the notion of security among architectures, designers and developers, and the lack of awareness of security in the SDLC, the techniques of process

improvement suggest finding better ways of incorporating security into software. This can be done by building security in such a way that security aspects are injected into the SDLC, and awareness should be created between stakeholders, developers, architects and developers—all of whom are involved in the software development process.

V. Traditional approaches for software security

Software security has been viewed as a set of firewall tools, encryption and testing that incorporate security into software development. A group of developers usually consider that a software shipment with firewall tools and authentication could be more than enough for securing software. Also, many software builders and security practitioners consider software security as an add-on feature to the software system. "Present software engineering practice in the industry does not lead to secure software at all" [34]. Unfortunately, for the last few years, many software practitioners and software builders consider software security as one of the quality features that need to be dealt with when product is shipped, or before deployment.

More recently, there is an incomplete view that has been suggested by security and industry experts, practitioners, and researchers about software security. In early 2002, an alert flag was raised about the importance of building secure software, by Trustworthy Computing Initiative [34]. In 2004, Michael Howard, a security expert at Microsoft, warned about this problem: "Few software developers follow security best practices to produce more secure code. Worse, they think of security after the fact. But it's a mistake to separate security consideration from the general software development process." [35] Many other security researchers have fought the traditional methods of software security,

such as Gary McGraw: "Software security is not just about building security functionality and integrating features!" He also clarifies that "just as you can't test quality into a piece of software, you can't spray paint security features onto a design and expect it to become secure." [3]

2.2 Software Development Lifecycles (SDLC)

The software development lifecycle is a methodology for the design and implementation of software solutions. Furthermore, a methodology may be defined as a formal approach to solving a problem, based on a structured sequence of procedures. The use of a methodology, therefore, ensures a rigorous process; it avoids missing any steps that could lead to compromising the end goal. It can therefore be argued that using a methodology increases the probability of success.

Peters et al. [36] define the software development lifecycle as "the period of time beginning with a concept for a software product and ending whenever the software is no longer available for use." Such a methodology represents the activities, their inputs and outputs, and any interactions during the software's lifecycle. Various software development lifecycle methodologies exist, including the waterfall, incremental, spiral, prototyping, evolutionary, object-oriented and agile models.

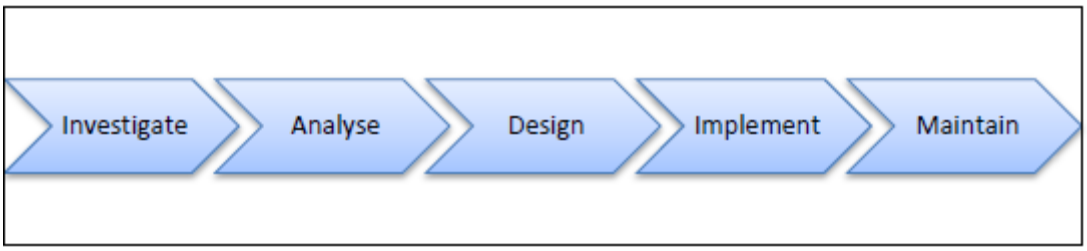

Figure 1: Typical phases of SDLC [37]

An important feature of the software development lifecycle is that it is a comprehensive method that encompasses five primary phases of software development, namely: investigation, analysis, design, implementation and maintenance, as depicted in Figure 1.

Although Figure 1 illustrates these phases as being applied in a sequential and linear manner, an iterative approach, as indicated in Figure 2, is more common. Since an iterative process is essentially circular in nature, each phase receives input from and provides output to another. At a high level, this circularity ensures the re-assessment of the quality of each artifact.

Figure 2: A Typical Iterative Cycle for Developing Software Solutions[37]

The five standard phases and their related activities are discussed in sub-sections 1, 2, 3, 4 & 5.

1. The Investigation Phase

During the investigation phase, the objectives, constraints and scope of the project are specified. It is during this phase that the problem definition and high-level requirements are established. The problem definition provides an initial description of the problem area, and it provides a firm foundation for the rest of the project. It typically takes the form of a written report and includes the current problems—as stated by the various stakeholders and interpreted by the developer—the objectives of the new system, and the scope and size of the project. The documentation produced during this phase requires user involvement. The problem definition report can be considered as the first stage in constructing the requirements specification document [38].

2. The Analysis Phase

Several basic activities of systems analysis need to be performed, whether developing a new application quickly or developing a long-term project. The analysis phase begins with a study of the documentation gained during the investigation stage. However, systems analysis is not a preliminary study, but an in-depth study of the detailed requirements. This phase produces a set of functional requirements that are used as the basis for the design of a new or improved software application. The functional and non-functional requirements are typically documented with text, use case diagrams, data-flow diagrams and other relevant figures, depending on the methodology followed.

3. The Design Phase

Whereas the systems analysis phase describes ‘what’ a software application should do to meet the identified requirements, the systems design phase specifies ‘how’ the application will accomplish these requirements. The systems design consists of various design activities that produce system specifications satisfying requirements specified in

the systems analysis phase. The objective of systems design is to describe the new software solution as a collection of modules or sub-systems. The systems design phase will indicate ‘how’ the new software application will be implemented, by providing all the necessary details, including data inputs, system outputs, processing steps and database designs. An important tool for software engineers is the Unified Modeling Language (UML) and its various diagrams, including use case, sequence and state transition diagrams. The output of this stage consists of a complete technical specification of the new software application [39].

4. The Implementation Phase

Once a new software solution has been designed, it must be implemented. During the implementation phase, the software application is physically built. This requires that program code is written and tested, and supporting documentation is produced including complex program listings, detailed test plans and instructions for operating procedures [39].

5. The Maintenance Phase

Once a software application is fully implemented and is being used in business operations, the maintenance phase begins. Software maintenance may be defined as the monitoring, evaluating and modifying of operational applications to make any desirable or necessary improvements [37].

- **Traditional Software Development**

The waterfall model is the oldest known SDLC model. It was first identified in 1970 as a sequence of activities from the requirements gathering until the coding stage. It describes a sequence of activities that begins with concept exploration and concludes

with maintenance and eventual replacement, as shown in Figure 3. Peters et al. [36] refer to it specifically as the forward engineering of software products.

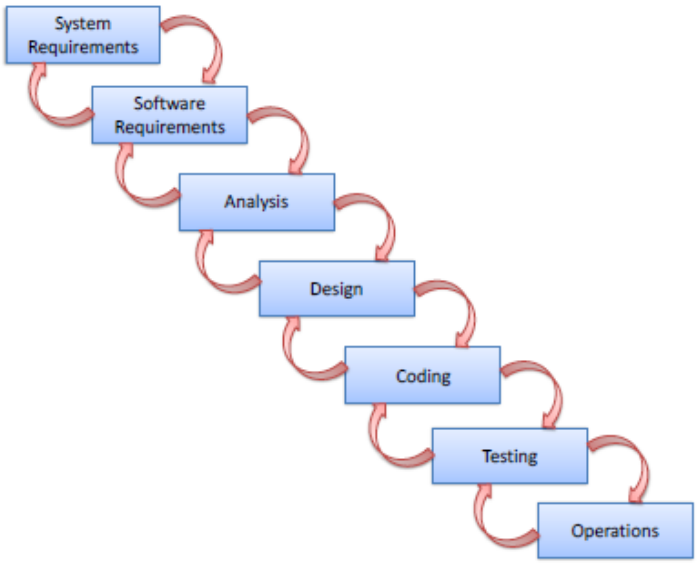

Figure 3: The Traditional Waterfall Mode

The application of the waterfall model, because of its inherent weaknesses, should be limited to those situations in which their requirements and their implementation are well understood. Large mainframe or complex client-server systems, and systems with highly complex technical requirements, may continue using this traditional approach. However, since it is not practical for most of the current applications which are running on highly networked PCs and workstations, a number of alternative software development models have emerged over recent years.

- **Alternative Software Development Models**

Various software development models have evolved from attempts to optimize the waterfall model. Modern software development processes are invariably iterative and incremental. This means that details are typically added in successive iterations allowing for changes and improvements to be introduced as needed. Incremental development

allows for a number of releases of software modules, thereby maintaining user satisfaction and providing important feedback to modules still under development [40].

There are many representations of the SDLC to choose from; all illustrating a logical flow of activity from the identification of a need through to the final software product. These methodologies use all the standards and procedures which will affect the planning, requirements gathering, analysis, design, development and implementation of a software system. Each SDLC model has its own strengths and weaknesses, and may therefore be better suited to certain types of projects within an organization. The expected size and complexity of the system, development schedule, and lifespan of a system will affect the choice of which SDLC model to use.

1. The incremental/evolutionary model

The product is said to evolve within the incremental/evolutionary lifecycle model because it consists of the planned development of multiple releases. Generally, increments become smaller and implement fewer requirements each time. It typically entails the continual overlapping of development activities and produces a succession of software releases. However, it can be costly if it is assumed that a current release is superseded by an improved version of the software later [41].

2. The spiral model

The spiral lifecycle model, introduced by Boehm in 1986, combines many good features of other software development models. These include the idea of baseline management (i.e., the documents associated with cycle phases), apparent in the waterfall model, the overlapping phases which are found in the incremental model, and early versions of a software application from the prototyping model. These software development models can be coupled with the spiral model in a natural way [42].

3. Extreme programming

Extreme programming is one of the earliest and most important of the Agile methodologies. It is a relatively new concept, but is in many ways an extension of the earlier work in prototyping and RAD. The main premise of XP is that the SDLC and its many alternatives are too large and cumbersome. Many of them provide good control but they typically end up adding complexity, taking more time, and slowing down programmers. XP simplifies the development process by focusing on small releases, similar to prototyping, that add value to the customer.

2.3 Security in the SDLC

Thinking about security at the early stages of software development by injecting security aspects could be achieved by looking for an already-existing software development model that provides security by design, or by trying to find some security principles that could be injected into each phase of the SDLC.

Tompkins et al. [43] found that as far back as 1985, inadequacies in the design and operation of computer applications were a frequent source of security vulnerabilities associated with information systems. This led them to state that "Security concerns should be an integral part of the planning, development and operation of a software application." Furthermore, they suggested that the SDLC methodology provides the structure to ensure that security safeguards are planned, designed, developed and tested in a manner that is consistent with the sensitivity of the information.

Tompkins et al. [43] found in the research that much of what needs to be done to improve security is not clearly separable from what is needed to improve the usefulness, reliability, effectiveness and efficiency of computer applications. However, they stress

that while security concerns should be integrated into the SDLC, steps should be taken to ensure that the appropriateness, adequacy and reasonableness of security safeguards be separately identifiable activities within each stage of the SDLC. This means that system planners, developers and users should accomplish a series of security-related actions throughout the SDLC. The process for incorporating security safeguards within an application, however, is not substantially different from the SDLC activities. Similarly, Jones et al. [44] state that to meet future demands, opportunities and threats associated with information security need to be "baked in" to the overall SDLC process. The reality is that information security is an afterthought for many organizations. This means that, most often, security is not an integral part of their business or information strategies, nor is it woven into their IT projects. Jones et al. [44] are concerned that traditional firewall systems have become less effective in preventing or detecting web-based attacks. They suggest that central to many successful system attacks currently are poorly developed systems and applications. Many of the security properties that are repeatedly outlined in government and other regulations, including accountability, unique user accounts and confidentiality, can be circumvented when software developers have not paid enough attention to security in the design, development, deployment and maintenance of their products. They argue that if the security considerations for systems were woven into the SDLC, and if the developers, project managers, and system architects were given adequate training, many of the security vulnerabilities that manifest themselves in software applications would never appear. Security plays an increasingly important role within systems' development. This can be attributed to the increase in the number of distributed applications. Breu [45] argue that security is a requirement that has to be

considered at all stages of development, and which needs particular modeling techniques to be captured.

2.4 The Important of a Systematic Literature Review

A systematic literature review is supposed to be a good guideline as a scientific method of research in the field of software engineering. It's also widely used and promoted in university environments. Kitchenham describes the importance of a systematic approach as: "Unless a literature review is thorough and fair, it is of little scientific value" [14]. In these terms, "systematic" means that others following the exact steps described in the article will achieve the same results and find the same resources as the article describes. The interpretation can be different, but it should not be possible to simply skip resources not fitting your thesis. In other words, the review can be reproduced by a third party and all they need is the description in the review itself. This intends to make sure that a research doesn't support only the preferred hypothesis of the researcher but also outlines the research that contradicts its own hypothesis. Another reason why I chose the systematic literature review is that in software engineering we find many studies covering different topics, but they often lack the scientific quality and reproducibility which we can find, for example, in medical research. The approach of a systematic literature review is to try to get more quality into the science of software engineering. A systematic literature review requires considerably more effort than a conventional literature review, but provides a much stronger basis for making claims about research questions [14].

During the last few years, a number of papers have focused on secure software development, some of which have carried out reviews and comparison studies on the issue. However, most of these reviews focused only on the secure software engineering at the requirements engineering phase of the SDLC and others concentrated only on special software development methodologies, such as Agile or XP. After performing preliminary searches aimed at both identifying existing systematic reviews and assessing the volume of potentially relevant studies, we can highlight a few works in a summary of a small group of security approaches in the security requirement of the requirements phase, such as [5]–[10], as well as a review of security approaches for specific domains such as web application[11]. There were also some reviews that investigated software security in specific development methodologies such as agile and XP [12] [13]. However, none of them performed a review focused on software security approaches that cover all stages of software development lifecycles in a systematic manner, and none of those reviews documented the systematic processes for selecting the initiatives (primary studies). Thus, there is a need to investigate the available security approaches and their limitations—as well as the stages in which these approaches are incorporated—in a systematic manner, to identify the gap in this area for further contribution by both the researcher and the practitioner.

Hence, an SLR was an appropriate research method for our research which aims to highlight the security approaches used for incorporating security concerns into software development lifecycles from the requirements gathering stage until maintenance and deployment, and identifies the possible limitations of doing so from the early stages of the SDLC.

CHAPTER 3

LITERATURE REVIEW

In this chapter we present a brief review of the related and current literature with respect to integrating security into software development lifecycles and the work that has been done in this area. Secure software development and the idea of building security as an integral part of the SDLC are discussed in Section ٣.1. Section 3.2 gives an overview of the existing work carried out so far in secure software development.

3.1 Secure Software Development

One of the key areas of concern is that of secure software development. It is important to note that the term ‘software’ is used and not that of ‘system’, since an information system is broadly defined as an organized combination of people, hardware, software, communication networks and data resources. Although the focus of this thesis is on the software aspect of an information system, it is understood that this cannot be studied in isolation, without any consideration of the other components [46]. Data comprise a critical asset to any organization; and they, therefore, need protection [46] . Software applications can be seen as the agents and processors of data. Although most organizations today have strong network perimeter controls in place, internally their applications and data are mostly left unprotected. Unfortunately, in many cases of software development, security is a mere afterthought [47]. According to Daud et al. [48], security typically goes unnoticed in the early phases of the software development life cycle. A good software engineering approach, however, is to consider security throughout

the software development lifecycle. In order to achieve this, it is necessary to adopt a process that incorporates all aspects of software development in secure software applications. Therefore, there exists a need to update and improve the current software development approaches. Process improvements should be added at every step of the software development lifecycle, regardless of the particular methodology chosen, to better focus on security issues.

Taylor and Azadegan [49] support this guideline, and state that: "Building secure systems requires incorporating security principles early and often throughout the software development life cycle." Software security should be an integral part of the development process; and it should be incorporated at every phase of the SDLC. Similarly, Microsoft supports the idea of injecting security into the SDLC "from the ground up", by adding suitable security checkpoints and touch points through the software development lifecycles.

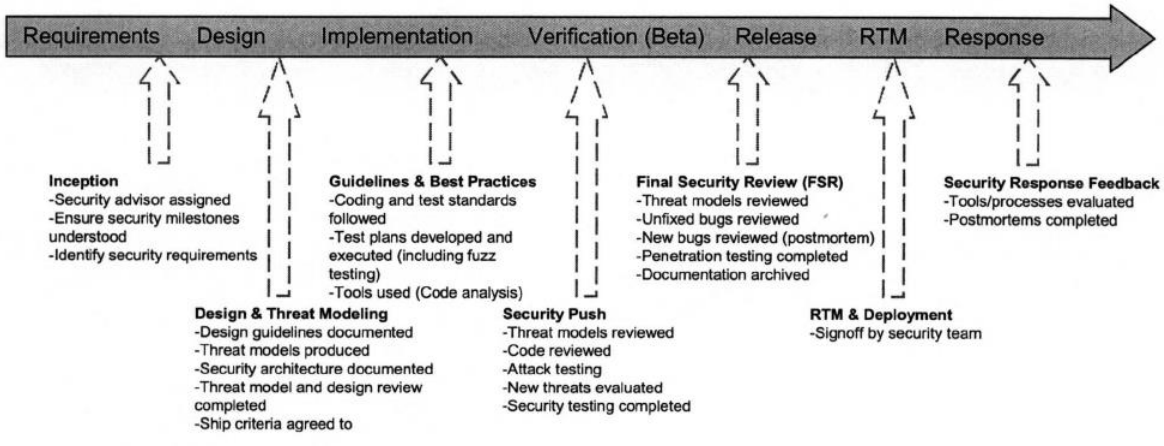

Figure 4 SDL Overview [50]

To address the need for secure software, Microsoft has adopted the Trustworthy Computing Security Development Lifecycle (SDL), shown in Figure 4. The SDL is intended to minimize the number of security vulnerabilities present in the design, coding and implementation of software, and to detect and remove these vulnerabilities as early in the lifecycle as possible. The need to consider security 'from the ground up' is a fundamental principle of secure software development [50]. Furthermore, OWASP has developed a set of CLASP best practices of software security. OWASP [52] states that: "To be effective, best practices of software security must have a reliable process to guide a development team in creating and deploying a software application that is as resistant as possible to security vulnerabilities." OWASP, therefore, recommends that the CLASP [52] best practices should form the basis of all security-related software development activities throughout the software development lifecycle. Also, McGraw [3] points out that security can be integrated into software development lifecycles and proposes seven touch points as depicted in Figure 5. These touch points are considered a small, manageable set of best practices for software practitioners to apply during software development, based on the artifacts they already produce. These software security best practices have their basis in good software engineering; they involve integrating security throughout the software development lifecycle.

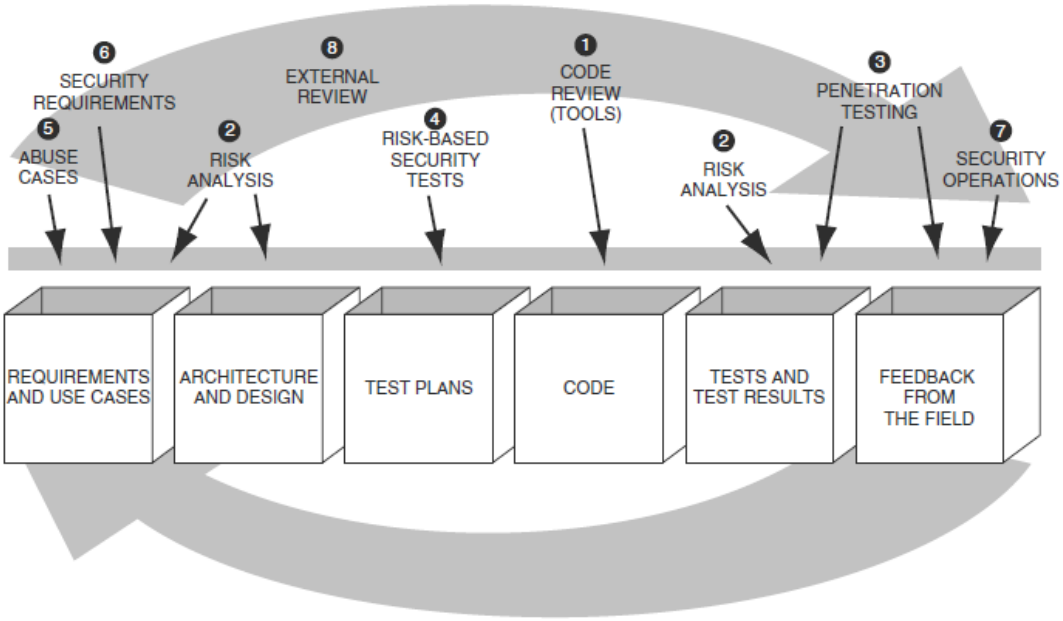

Figure 5 Seven TouchPoints for Software Security[3]

A number of researchers have argued the need to consider security from the early stages of software development lifecycles, from the early requirements until coding and maintenance [51]–[54][55]–[57]. One of the popular models for integrating security into the SDLC in the requirements phases is misuse case, which is based on the use case approach. Use cases document functional requirements of a system by exploring the scenarios in which the system may be used. Scenarios are useful for eliciting and validating functional requirements [51], but are less suited for determining security requirements which describe behaviors not wanted in the system. Similar to anti-goals [52], misuse cases are a negative form of use cases and thus are use cases from the point of view of an actor hostile to the system [58]. They are used for documenting and analyzing scenarios in which a system may be attacked. Once the attack scenarios are identified, countermeasures are then taken to remove the possibility of a successful

attack. Figure 6 shows some of the use cases and misuse case of a bank account system. Use cases are represented as clear ellipses while misuse cases are represented with the shaded ellipses. The <<threatens>> stereotype implies that the given misuse case is a threat to the satisfaction of the requirements of the corresponding use case. The notation we use for misuse cases is based on requirements of the engineering process proposed by Sindre and Opdahl [51].

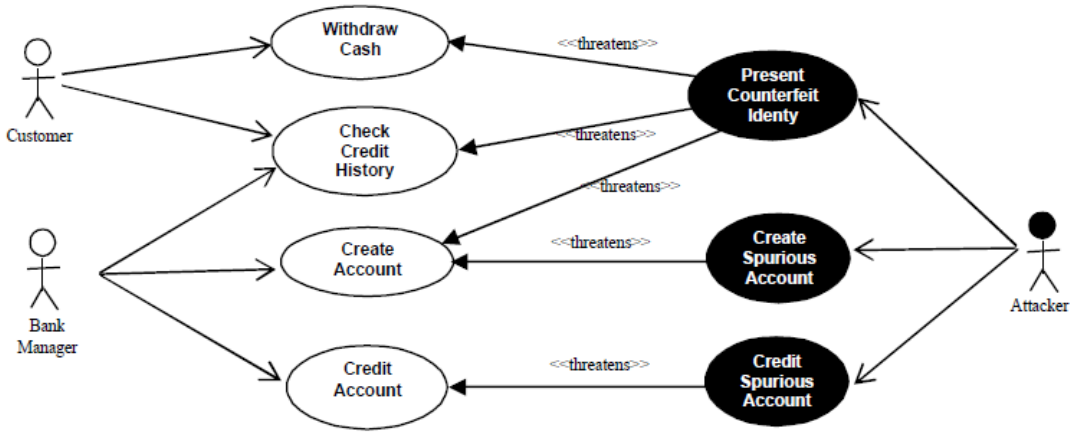

Figure 6 Use and Misuse case of a banking system[51]

Another use case approach that deals with security requirements is abuse case [59]. This approach uses UML use case diagrams for presenting unwanted behavior of a piece of software. In this approach, the abuse case model is developed and used to present the harmful interaction between a normal user (an actor) and the abuse cases. Also, many approaches extended the normal UML for modeling software security such as SecureUML and UMLsec [13] [14] among others. This is because UML does not

originally cover non-functional characteristics (including security) in an explicit way. It is possible to analyze and represent vulnerabilities in the target system, and the vulnerabilities can be mitigated from the viewpoints of structure and dynamic behavior. SecureUML [56] focuses on modeling access control policies and how these (policies) can be integrated into a model-driven software development process. It is based on an extended model of role-based access control (RBAC) and uses RBAC as a meta-model for specifying and enforcing security. RBAC lacks support for expressing access control conditions that refer to the state of a system, such as the state of a protected resource. Addressing this limitation, SecureUML introduces the concept of authorization constraints. Authorization constraints are preconditions for granting access to an operation. UMLsec [57] is also an extension of UML which allows an application developer to embed security related functionality into a system design and to perform security analysis on a model of the system to verify that it satisfies particular security requirements. Security requirements are expressed as constraints on the behavior of the system, and the design of the system may be specified either in a UML specification or annotated in source code.

Secure software does not mean software that is entirely hack-resilient, with no vulnerabilities. Nor does it mean zero-defect software, since such software does not exist. Secure software is software designed with security in mind, developed with appropriate security controls and deployed in a secure state [46]. A common misconception is that secure software is all about technology or code security. While writing secure code is a critical component of software development, there is a lot more to consider. A secure

software development lifecycle requires the convergence of policy, processes and people [46]. These are described as follows:

- Policies, standards, best practices and procedures should be formulated to establish a secure software development methodology.
- Secure software processes must ensure the incorporation of security into the software development lifecycle, including secure programming and software risk management.
- People are vital to any organization; they need to be educated in protecting an organization's data and in developing secure software.

3.2 Exiting Works

This section presents a review of the key studies conducted on the topic of secure software development and integration security in various stages of software development lifecycles. The objective is to summarize and discuss the results of each study, which gives better understanding of the problem in context.

Hadavi et al. [6] have focused their review only on security requirements engineering by reporting state-of-the-art and research challenges in security requirements. The current knowledge in security requirements and threat modeling has been a synthesis and has provided the first step for integrating security requirements in the software development process. Furthermore, based on the type of activities and methods, they presented more than twenty-three research directions and classified them into five categories.

Salini and Kanmani [60] have reviewed the literature to compare and analyze different methods of security requirements engineering (SRE). The authors also presented a view on security requirements types and issues. Finally, the important activities of security requirements engineering have been presented and the identified SRE methods are compared based on these activities.

D. MU et al. [61] have focused their research work on security requirements engineering process and methods. Additionally, the authors examined the compatibility of these process with respect to model-driven engineering (MDE) and risk analysis (RA). This evaluation could help in understanding and selecting the SRE processes and methods.

Du et al. [62] have performed a literature review and analysis of software security requirements engineering development methodologies. The results have been reported and investigated with respect to the literature source, research community, publication year, research region, security understanding, activities and methods types. Furthermore, they divided the identified approaches based on different categorizations: technologically driven, process oriented and others, such as the extension of the Unified Modeling Language (UML).

Khan and Zulkernine [8] conducted a comparative study that presents a complex survey on the requirements and design phases of secure software development. Different

activities that should be done in the requirements and design phases have been identified, and comparisons between different modeling languages and process are made in this study. This study provides the developer with guidelines that help him with selecting the best fitting method for building secure software.

Tondel et al. [10] have conducted a related study wherein they focused on the tasks recommended in the requirements phase. In this study the authors have surveyed concrete techniques for eliciting security requirements. Nine techniques have been surveyed, which are presented as a series of well-defined steps that collectively lead to the elicitation of security requirements.

Fabian et al. [5] have introduced a conceptual framework for security requirements methods. The aim was to compare and evaluate current security requirements engineering approaches, such as the Secure Tropos, common criteria, MSRA, and SREP, as well as methods based on UML and problem frames. The authors have reviewed and assess the methods based on the proposed criteria and have classified the identified approaches into six categories (multilateral, UML-based, goal-oriented, problem-frame based, risk-oriented, Common Criteria-based). Furthermore, they systematically discussed these approaches in four sections (i.e., general description, scope, validation and quality assurance, and relation to the conceptual framework).

Different studies have been conducted to compare and analyze various software security approaches for specific security policy such as RBAC modeling and documentation. Matulevîcius and Dumas [63], investigated and analyzed two modeling

languages, namely SecureUML and UMLsec, that could help define security policies through the role-based access control mechanism. This investigation will help the modeler in selecting the appropriate technique for RBAC analysis. Also in a recent study, Raspotnig [64] reviewed and compared techniques for safety and security requirements.

Previous systematic literature reviews such as that by Ghani and Yasin [12] have focused on security adoption inside an extreme programming model and have explored the models or frameworks that relate to secure XP methodologies. It further investigated the compatibility of the extreme programming model with software security engineering. Similarly, F. Roeser [13] conducted a systematic literature review to identify security practices that have been developed and adopted to fit in with Agile methodology. A quick review was done to investigate how successful these practices could be if implemented using Agile methodology.

A systematic literature review conducted by Mellado et al. [9] focused on security requirements engineering. The review considers studies that have incorporated security only at the requirements stage of information system development, without paying much attention to security across the entire software development lifecycles. Musa [11] used a systematic review to investigate the various security development models used to secure web applications, security approaches used in the process, and the stages in the development model in which the approaches or techniques are emphasized. Moreover, one recent research involving a systematic mapping study was conducted by Dasanayake [65] to identify various aspects of addressing concerns throughout the software

development lifecycles and to study most considered concerns and their variations in the SDLC. The considered concerns in this study included security, reliability, maintainability and performance.

Our work is on alignment with the previous study in identifying the approaches for secure software development from the requirements gathering stage until the maintenance stage. Nevertheless, and to the best of our knowledge, no SLR has been done in this area before, which covers the entire software development lifecycles (SDLC). Software security is a somewhat mature area and both industry and researchers are attracted to it. In the field of industry, this topic is of interest in order to avoid severe losses due to the consequences of insecure software. Similarly, this topic is attractive to the researcher as this area is new and there are many gaps that need to be discussed in a systematic way for enhancement and innovation. Both industry and academia can benefit from this thesis. In academia, this thesis can provide an understanding of software security and the different approaches that can be incorporated in different phases of SDLC, as well as the security activities that can be aligned with normal development activities. Also, many interested researchers and practitioners can benefit from this work by contributing to this area through exploiting the gaps and the limitations identified in this work. For example, one researcher may propose an approach that helps security testing as there are no more approaches identified at this stage. Furthermore, most of the approaches in the coding stage do not cover more vulnerabilities, so new interested researchers can contribute to this area by enhancing these approaches to cover more vulnerabilities. Moreover, the academic institutions can benefit from this work by

updating their programs in computer science and software engineering majors by teaching their developer the appropriate security practices and guidelines that help them in building secure software. Finally, in industry, this thesis can help in choosing one or more of the security approaches that can be injected in the appropriate stage to build secure software and apply these approaches in the real industrial environment to see their effectiveness.

The main contribution of this study is to add to the body of knowledge of both disciplines: security and secure software development lifecycles. Furthermore, we are going to assist software development organizations in better understanding the limitations of existing software security approaches used in the software development lifecycle and to provide other researchers with a firm basis on which to develop of the new software security approaches. Moreover, comprehensive mapping studies of the secure software development approaches and in which stage these approaches are emphasized, will be conducted. Additionally, the most active researcher in software security and the main venue of publication of software security will be identified. Finally, the gap and limitations in the existing approaches will be studied and analyzed.

CHAPTER 4

RESEARCH METHODOLOGY

We followed two complementary methods to achieve maximum coverage and to make our research more comprehensive as depicted in Figure 7. In order to address our research questions, we applied the systematic literature review (SLR) and snowballing. In the first method, we identified software security approaches and its limitations that are used for incorporating security concerns into software development lifecycles (SDLC) and the stages in which these approaches are emphasized via a systematic literature review. The most active researcher and publication venue are also identified. We then used snowballing as a second method to find the possible limitations and challenges for incorporating security concerns into the SDLC. We discussed each of the research methods in detail in the following sections. Section 4.١ explains the whole SLR process, which includes developing an SLR protocol, cleaning and processing the findings via initial and final study selection, validation and filtration using quality assessment techniques, and data synthesis and proofreading. Section 4.٢ explains in summary about the snowballing method for finding the possible limitations of incorporating security concerns and existing software security approaches in the SDLC.

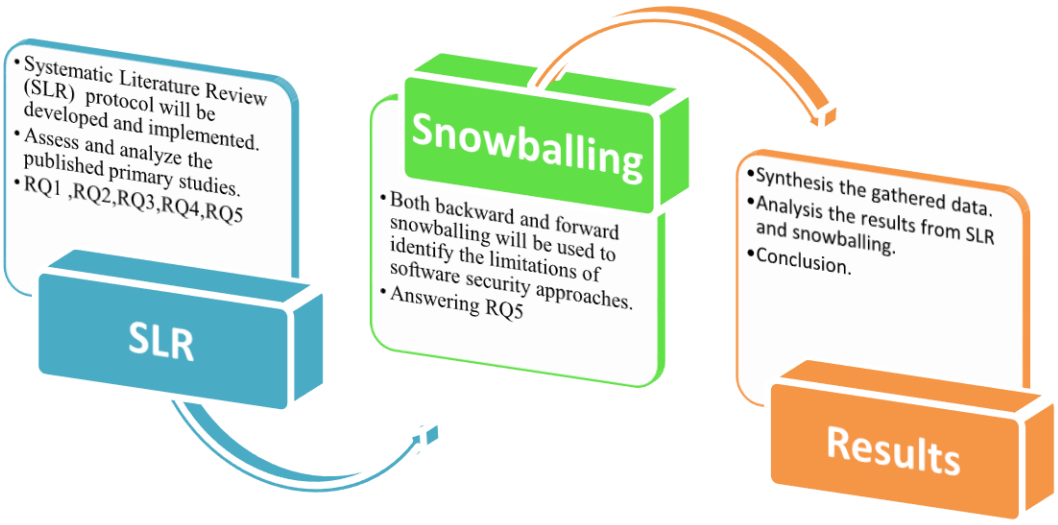

Figure 7 Research Methodology

4.1 Systematic Literature Review (SLR)

We have followed the SLR guidelines proposed by Kitchenham and Charters for performing an SLR for data collection, since it is a well-defined and rigorous method to identify, evaluate and interpret all the relevant studies regarding a particular research question, topic area or phenomenon of interest. A systematic review is a defined and methodical way to summarize the empirical evidence concerning a treatment or technology, to identify missing areas in current research or to provide background in order to justify new research. Systematic literature review requires considerably more effort than conventional literature review, but provides a much stronger basis for making claims about the research questions [14]. Hence, an SLR was an appropriate research method for our research, which aims to highlight the security approaches used for

incorporating security concerns into software development lifecycles from the requirements gathering stage until maintenance and deployment, and identifies the possible limitations of doing so from the early stages of the SDLC.

A systematic literature review protocol was written to provide the details of all steps that we have followed in our study; the major steps are described as the following:

- Constructing a search strategy and then performing the search for relevant studies.
- Study selection process.
- Apply quality assessment for the selected study.
- Conducting data extraction, mapping then analysis of the extracted data.

The details of these summarized points are depicted in the next figure and will be described in the next sub-sections.

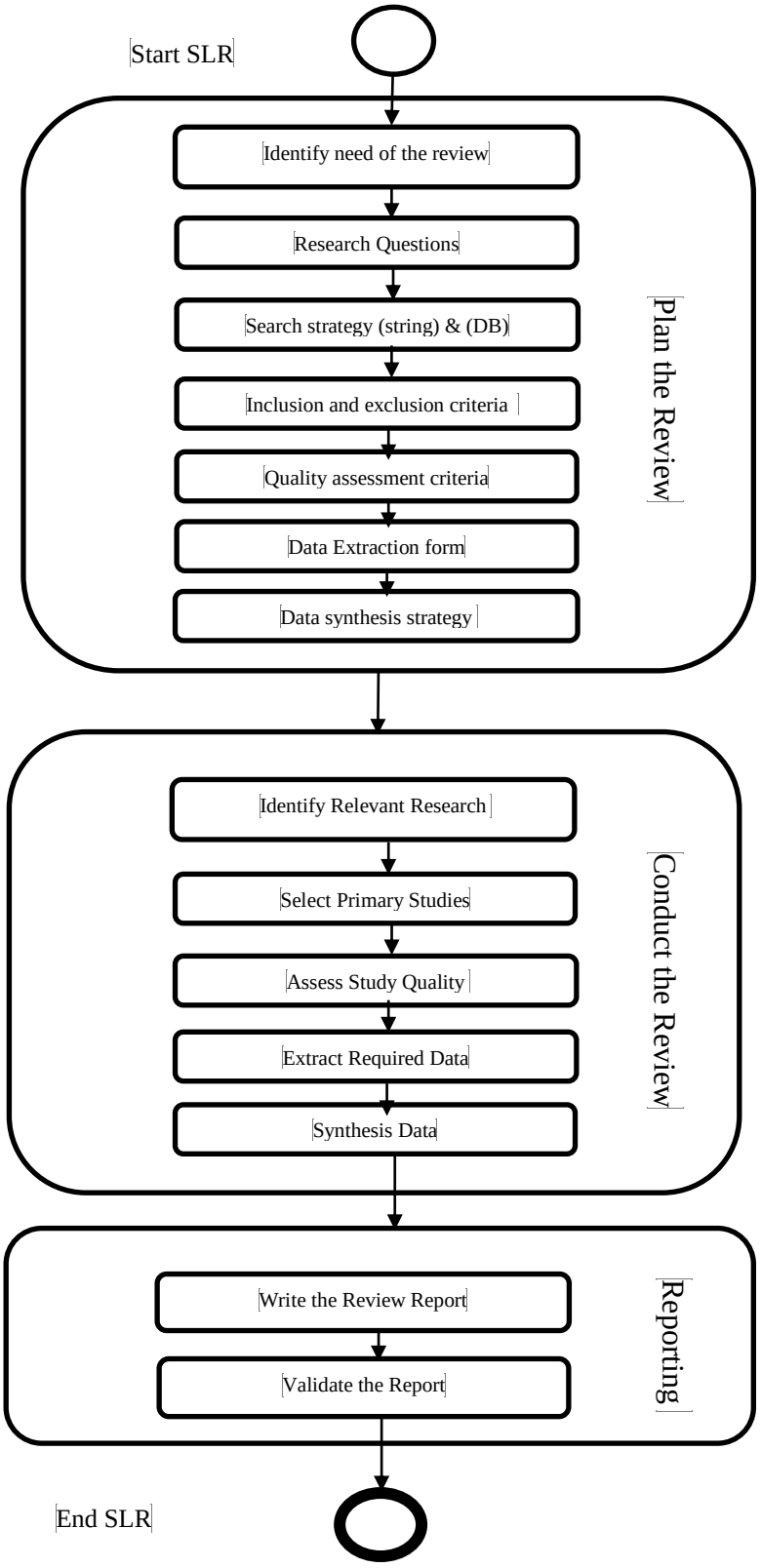

Figure 8 selection process

RQ1: What approaches are available for secure software development?

RQ2: In which stage of the software development lifecycles are the software security approaches incorporated?

RQ3: Which researchers are most active in software security?

RQ4: What are the main venues for publications on software security?

RQ5: What are the limitations of each of the software security approaches used in the software development lifecycles?

4.1.1 Search Strategy

The strategy used to construct search terms is as follows:

a) At the beginning we have derived the major search terms from the research question by identifying the population, intervention and outcome.
b) We then identified alternative spelling and synonyms for the derived major terms to ensure that we don't miss any related study.
c) We have verified and checked the keywords in relevant papers.
d) We have used Boolean operators: "AND" to concentrate the major terms, and "OR" to concentrate synonyms and alternative spelling, where the database allows.
e) Finally, we have integrated the search string into a summarized form, if required.

Result of a

The following details of the population, intervention, outcome of relevance and experimental designs of interest to the review will form the basis for the construction of suitable search terms later in the protocol.

Population: is the application area, in this context it is the software security

Intervention: the existing approaches for secure software development lifecycles

Outcome of relevance: secured software development lifecycle approaches, secured SDLC processes, models of security.

Experimental design: SLR, empirical studies, case studies, theoretical studies, expert observation and expert opinions.

The above details for the RQ1 and the result of RQ1-4 will come automatically from the identified studies in RQ1. However, snowballing has been used for answering RQ5 as we mentioned earlier in this chapter.

Result of b

Approach:

"secure development methodologies" OR "security principles" OR "security standards" OR "security practice" OR "framework" OR "approach" OR "technique" OR "model" OR "method" OR "tool" OR "development “practices" OR "development guidelines" OR "best practice" OR "engineering process" OR "security activities" OR "development cycle" OR "development guideline" OR "development principle" OR "development

procedure" OR "development approach" OR "development lifecycle" OR "development model" OR "development framework" OR "development practice"

Software security:

"software quality " OR "software safety" OR "information security" OR "software vulnerability" OR " application security" OR" "secure" OR "insecure" OR "software security" OR "confidentiality" OR" authorization" OR "authentication" OR "integrity" OR "access control" OR "secure system" OR "secure application" OR "secure software" OR "authentication" OR "privacy" OR "access control" OR "confidentiality" OR "secrecy" OR "integrity" OR "availability" OR "auditability" OR "authorization" OR "threat model" OR "attack model" OR "intrusion detection" OR "information flow" OR "encryption"

Software development lifecycles:

"software development" OR "systems development" OR "software development lifecycle" OR "SDLC" OR "software development process" OR "software development activities" OR "early development stages" OR "engineering process" OR "software development methodologies" OR "application development process" OR "secure IS development"

Result of c

Approach:

"development guideline" OR "development principle" OR "development procedure" OR "development approach" OR "development lifecycle" OR "development model" OR "development framework" OR "development practice"

Software security:

Software vulnerability OR Application security OR Secure OR Insecure OR Software security

Software development lifecycles:

"secure software development" OR "secure systems development" OR "secure software development life cycle" OR "systems development lifecycle" OR "SDLC" OR "software development process" OR "secure IS development" OR "software development lifecycles"

Result of d

("development guideline" OR "development principle" OR "development procedure" OR "development approach" OR "development lifecycle" OR "development model" OR "development framework" OR "development practice") AND (software vulnerability OR application security OR secure OR insecure OR software security) AND ("secure software development" OR "secure systems development" OR "secure software development life cycle" OR "systems development lifecycle" OR "SDLC" OR "software development process" OR "secure IS development" OR "software development lifecycles")

Based on the available access, the following digital libraries were used:

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

Since these libraries differ in their search mechanisms and capability, we tailored our search string accordingly.

4.1.2 Publication Selection

4.1.2.1 Inclusion Criteria

The inclusion criteria we have identified to determine which part of literature returned by the search string would be used for data extraction.

- Studies that are reported in English language only.
- Papers published in any of the primary or secondary resources mentioned previously.
- Studies focused on answering our research question.
- Source is a research paper, proceeding, book chapters, lecture note in computer science or journal article.
- Studies focused on incorporating security form the beginning of software development (i.e. from requirements gathering until deployment stage), by proposing security concerns or approaches for building security in.

- Any study that focused on some kind of approaches and techniques to follow in order to carry out security activities during the phases of software development.

4.1.2.2 Exclusion Criteria

- Studies that were not relevant to the research questions.
- Manuscripts written in non-English language were excluded.
- Studies with poor English were excluded as the sentences may cause ambiguity or exposes conflicts of ideas.
- Graduation projects, Master's thesis and PhD dissertations were excluded as they tend to be much more focused and there is no evidential proof of any review.
- The paper only mentions security as a general introductory term.
- Studies focused on software security in the later stages after deployment.
- Studies focused on security through penetrate and patch or that concentrated on secure software using external devices such as firewalls and other protection mechanisms.
- Studies that focused on information security mechanisms such as encryption and decryption.

4.1.3 Selection Primary Sources

The selection process had mainly two phases as planned in the review protocol: an initial selection from the search results based on reading the title and abstract of the paper; then by final selection from the first step by reading the full paper. These processes are depicted in Figure 9.

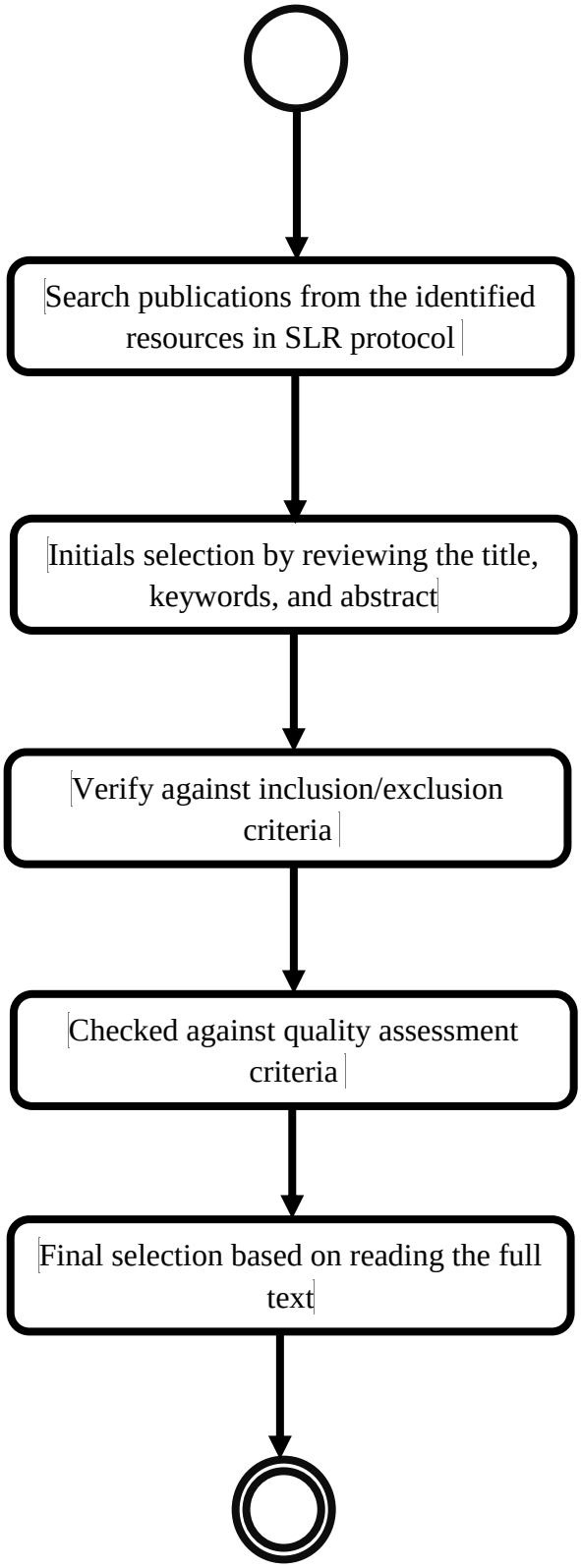

Figure 9 Selection Process

The total number of results retrieved after inputting the search terms in the electronic databases are shown in Table 1 Primary Studies Selection from different resources. After the initial round of screening by reading the title and abstract, about 184 studies belonging to five different electronic research databases were selected. After the full text readings in the second screening, 118 primary studies were finally selected, which met our inclusion and quality criteria.

Table 1 Primary Studies Selection from different resources

Resources	**Total Result**	**Initial Selection**	**Final Selection**
IEEEXplore	1880	75	53
ScienceDirect	506	35	15
SprinerLink	437	39	25
ACM	140	28	24
John Wiley	93	7	1
Total	3056	184	**118**

4.1.4 Quality Assessment

In addition to the inclusion/exclusion criteria, the quality of each primary study was assessed by the quality checklist for quantitative studies. The quality assessment was performed after we finished the final selection of publications; for any paper to pass the initial phase, a quality assessment was done. We have to assess the quality of the literature selected after final selection for quality. The quality assessment activity for the

relevant literature was carried out at the same time during the extraction of relevant data so as to ensure that a valuable contribution was made to the SLR. We will detail a quality assessment checklist that will provide the means to quantitatively asses the quality of the evidence presented by these studies. However, these checklists are not meant to be a form of criticism criteria, as such will be documented. These quality criteria were prepared as shown in Table 2. Each question in the quality checklist was answered with 'Yes' or 'No', and marked by 1 and 0 respectively. The final score of the study ranged from 0 to 6, where 0 is the lowest score, representing lower quality, and 6 is the highest score, representing high quality studies, according to our definitions. A threshold value for excluding a study from the review was set at 3 points. Since the lowest score for the study was 4, all the studies were included on the basis of the quality checklist.

Table 2 STUDY QUALITY ASSESSMENT TABLE

Criteria	Notes
The approach is explained sufficiently.	Yes =1 No =0
Evidence of the approach is documented.	Yes =1 No =0
Does the study state clear, unambiguous aims of the research?	Yes =1 No =0
Is there any empirical evidence on the findings?	Yes =1 No =0
Is the paper well/ appropriately referenced?	Yes =1 No =0

Is the paper legible and well written?	Yes =1 No =0

4.1.5 Data Extraction

After the final selection of primary studies, depending upon the quality assessment criteria, we have to start with the data extraction phase of the systematic literature review process. We used the data extraction form to extract the data. The data was extracted by a single reviewer, who was alone responsible for data extraction, and then assessed by a PHD supervisor in a random manner. Table 3 represents the data extraction form which was used for the purpose of extracting relevant data from primary studies.

The data extracted from the primary studies was saved as a Microsoft Word document in <paper id> _ <author name> _ <Year of publication>, while a tool called *Mendeley* was used for reviewing and controlling the selected primary study.

Table 3 Data Extraction Form

Data Item	Value	Supplementary Notes
Study Information Data		
Paper ID		
Title		
Date of publication		

Author		
Year of publication		
Reference type	Journal/Conference/Thesis/Unpublished	
Geographical location		
University/organization		
Publisher		
Methodology/ Type of study	SLR/Interview/Case Study/Report/Survey	
Data Relevant to Answering Research Questions		
Security Approach		
SDLC phase		
Venues for Publication		
Active Researcher		

4.1.6 Data Synthesis

After the extraction of data we used the data synthesis form as shown in Table 4, to summarize and compile the extracted data from the primary studies so as to answer each of the research questions. This form helps to carry out various types of statistical analyses so as to draw a conclusion.

Table 4 Data Synthesis Form

RQ1 - 4						
Security Approach	*SDLC Stage*	*Venue of Publication*	*Type of the Study*	*Geographical Location*	*Reference Type*	*Active Researcher*

Due to the nature of the research questions we are going to synthesis the extracted data by checking the most frequent approaches that were used for incorporating security concerns into the SDLC and the most frequent phases, where the security approaches are emphasized. This categorization will help in identifying the most neglected stage in terms of security so the new room will be opened for further research.

Additionally, we are going to analyze studies based on the countries where they were conducted, the active researchers, and on the publication venue where the publication channel of the study.

4.2 Snowballing

With our search string used in the SLR we could not identify enough number of papers for RQ5. So it was decided to use alternative search, i.e., snowballing.

In addition to the searches in the databases using search strings, snowballing from the list of references of the identified articles was used as another method in this research (i.e., to identify additional relevant articles through the reference lists of the articles found

using the search strings and articles identified through manual search using Google Scholar to answer the **RQ5**. Both backward snowballing from the lists of references and forward snowballing which is finding the citations to the papers, were included in this research.

The snowballing search method [66] can be summarized in three steps: 1) Start the searches in the leading journals and / or the conference proceedings to get a starting set of papers. 2) Go backward by reviewing the reference lists of the relevant articles found in step 1 and step 2 (iterate until no new papers are identified); and 3) go forward by identifying articles citing the articles identified in the previous steps. Based on Webster and Watson [66] as well as Wohlin [67], the starting point for the backward snowballing research approach is the analysis of main contributions to the topic. Thus, we identified our starting sets of papers using the common primary studies that have been identified through databases automatic search using a search string that presented common approaches found in the literature, such as misuse case, Secure Tropos, KAOS, UMLsec, secureUML and static and dynamic analysis among others. Also, the systematic literature reviews that have been identified through manual search using Google Scholar are included in the starting sets of papers. A manual search through Google Scholar using terms such as “challenge" OR "limitation" OR "problem" OR "difficulties" OR "trouble" OR "issue" OR "weakness” and integrated with terms such as “secure software development”, “software security”, “secure early stage of software development stages” OR “secure information system development” were used to find starting sets. The snowballing procedure is outlined in steps in Figure 10[67].

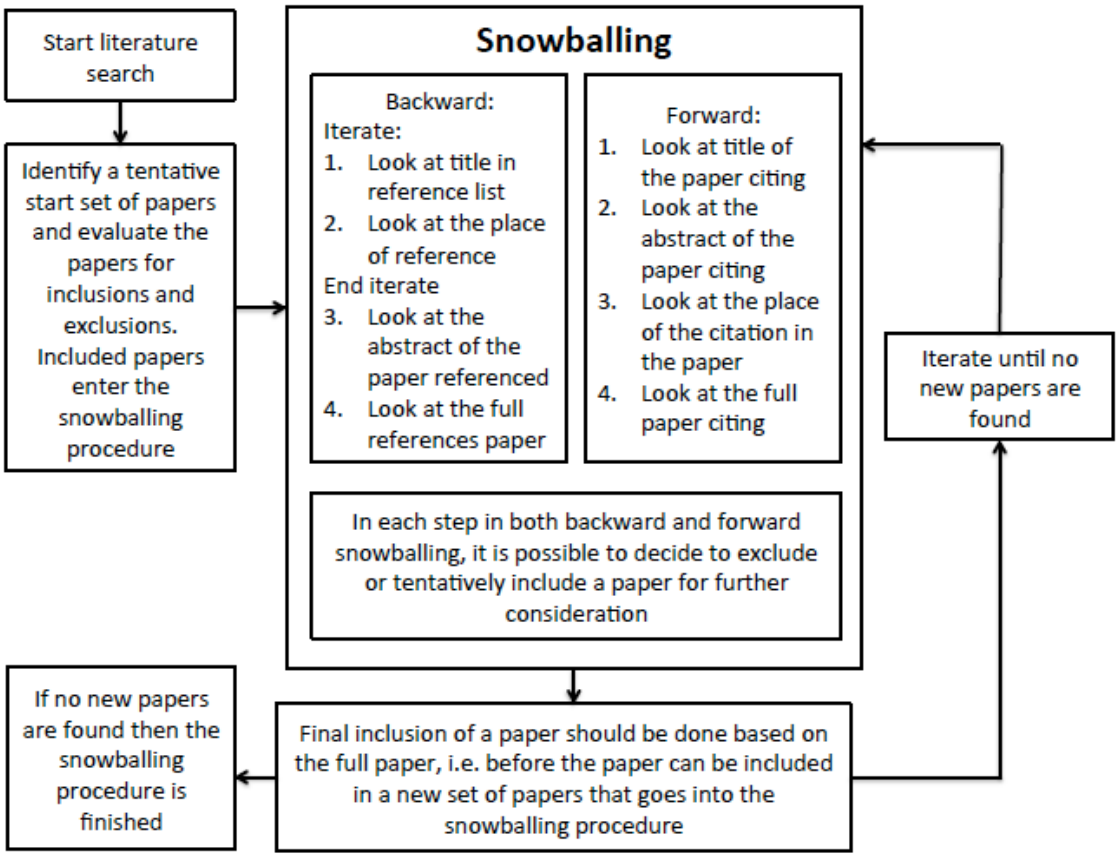

Figure 10 The Snowballing Process [67]

We retrieved approximately fifty-three papers using the snowballing process. After applying the selection criteria, we selected forty-seven papers for data extraction and analysis. Total 165 primary studies from SLR and Snowballing were selected that met our inclusion criteria and quality assessments are shown in Table 5. For the others, meaning this articles doesn't include to any of the mention digital library and it comes from different journals and conferences.

Table 5 Results (SLR + Snowballing)

Resources	Total Result(SLR)	Initial Selection(SLR)	Final Selection(SLR)	Snowballing	Duplication
IEEEXplore	1880	75	53	18	2
ScienceDirect	506	35	15	3	1
SprinerLink	437	39	25	6	1
ACM	140	28	24	5	1
John Wiley	93	7	1	-	0
Others	-	-	-	15	0
Total	3056	184	**118**	**47**	**5**
Total			**165**		

CHAPTER 5

RESULTS AND ANALYSIS

In this chapter we present the results and analysis from our two-phased research methodology. Section 5.1 explains the findings from the SLR that answer our research questions mentioned in the protocol. Section 5.2 answers the missing articles regarding to RQ5, which aimed to identify the limitations and challenges of incorporating security into software development lifecycles.

5.1 Systematic Literature Review (SLR) Results

This section presents the initial SLR-based literature survey results. The total number of results retrieved after inputting the search terms in the electronic databases is shown in Table 1**.** After an initial round of screening by title and abstract, about 184 studies belonging to five different electronic research databases were selected. After full text readings in the second screening and the application of inclusion and exclusion criteria, about 118 primary studies were finally selected. We analyzed each publication and extracted about 54 relevant approaches for incorporating security into different stages of SDLC. These approaches have been categorized (for better understanding) mainly into seven main categories shown in Table 6. Also, the list of 54 identified approaches for injecting security into SDLC and the phases in which this approach are incorporated shown in Table 7**. (For more details of these security approaches please see Appendix A).**

Table 6 Approaches Categorization

Group	Approach
Reverse approach ***(****Consider security requirements in a reverse way, e.g. identifying problems or attacks that may subvert the security of software systems****).***	Abuse frame
	Misuse cases
	Abuse case
	Essential Use case
Process Oriented ***(****Proper steps, procedures activities to guide the participants.)*	SREF
	Apvrille and Pourzandi
	Van Wyk and McGraw
	Microdoft SDL
	Software Security Assessment Instrument (SSAI)
	S2D-ProM
	SREP
	ISDF
	SQUARE
	CLASP
	AEGIS
UML-based approaches ***(*** *Approaches that make use the Unified Modeling Language notations)*	UMLintr
	Georg-AO
	Gomaa-UML
	YU-AC
	FDAF
	Kim-Access Control
	Mariscal-AC

	Medina-DB
	PbSD
	UMLsec
	SecureUML
	UML state charts
	Hoisl-SOA
	UML- AC
	UMLS
Notations ***(****Specify and present security specification (security properties, attack specification, security requirement) using new proposed notations.)*	ADM-RBAC
	AMF
	Xu-Petri
	Giordano-Access Control
	Buyens-LP
	SECTET
	AsmLSec
	AsmL
	SecureSOA
Vulnerabilities-Mitigation approaches ***(****Dealing with common security vulnerabilities in the coding phase such as BOF, XSS, SQLI ...)*	Static analysis
	Dynamic analysis
	Hybrid analysis
	Secure programming
	Program transformation
	Patching
Goal-oriented Approaches	KAOS

Extended for goal oriented modeling approaches that focuses on describing both organizational environment of a system and a system itself. Also, it extended for specifying the anti-goal and constrains of the systems.	SecureTropo
Others *(specific to certain technology or development methodology)*	HTTPUnit
	Vela-DB-XML
	SRS-Tool
	Agile Security Framework(ASF)
	STS-Tool
	Gupta- Framework
	FDD

Table 7 List of Security approaches

#	Approach	Phase	References	Freq.
1	Abuse Frame	Requirement	[68]	1
2	SREF	Requirement	[69] [70]	2
3	Apvrille and Pourzandi	Across	[54]	1
4	Van Wyk and McGraw	Across	[71]	1
5	Microdoft SDL	Across	[50] [72]	2
6	Software Security Assessment Instrument (SSAI)	Across	[73]	1
7	S2D-ProM	Across	[74]	1
8	Misuse cases	Requirement	[51] [75]	2
9	Abuse case	Requirement and Design	[59]	1
10	UMLintr	Requirement	[76]	1
11	AsmLSec	Requirement	[77]	1
12	ADM-RBAC	Design	[78]	1
13	AMF	Design	[79]	1

14	Georg-AO	Design	[80]	1
15	Gomaa-UML	Design	[81]	1
16	SecureSOA	Design	[82] [83]	2
17	UMLsec	Design	[84] [85] [53] [57] [55] [86]	3
18	Xu-Petri	Design	[87]	1
19	YU-AC	Design	[88]	1
20	KAOS	Requirement	[52]	1
21	HTTPUnit	Requirement, design , coding	[89]	1
22	Dynamic analysis	Coding	[90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [101] [102] [103] [104] [105] [106]	17
23	Static analysis	Coding	[107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] [118] [119]	13
24	Hybrid analysis	Coding	[120] [121] [122] [123] [124] [125]	6
25	Secure programming	Coding	[126] [127] [128] [129] [130]	5
26	Program transformation	Coding	[131] [132] [133] [134]	4
27	Patching	Coding	[135] [136] [137] [138]	4
28	SREP	Requirement	[139] [140] [141]	3
29	SecureTropo	Requirement, Design , Coding	[142] [143]	2
30	FDAF	Design	[144] [145]	2
31	Giordano-Access Control	Design	[146]	1
32	Kim-Access Control	Design	[147]	1
33	Mariscal-AC	Design	[148]	1
34	Medina-DB	Design	[149] [150] [151]	3
35	PbSD	Design	[152] [153]	2
36	Vela-DB-XML	Design	[154]	1
37	SecureUML	Design	[56] [155] [156]	3
38	UML state charts	Requirement	[157]	1
39	SRS-Tool	Requirement	[158]	1
40	SECTET	Design	[159] [160]	2

41	Buyens-LP	Design	[161]	1
42	Hoisl-SOA	Design	[162]	1
43	UML- AC	Design	[163]	1
44	UMLS	Design	[164]	1
45	Agile Security Framework(ASF)	Across	[165]	1
46	ISDF	Across	[166]	1
47	EUC	Requirement	[167]	1
48	STS-Tool	Requirement	[168]	1
49	Gupta- Framework	Requirement	[169]	1
50	SQUARE	Requirement	[170] [171]	2
51	CLASP	Requirement	[172] [173]	2
52	AEGIS	Requirement	[174]	1
53	AsmL	Requirement	[175]	1
54	FDD	Requirement - Design	[176]	1
Total				**118**

5.1.1 Approaches Frequency Analysis

Table 8 Approaches Freq. Analysis

#	Approach	Freq.(118)	%
1	Dynamic analysis	17	14.41
2	Static analysis	13	11.02
3	UMLsec	6	5.08
4	Hybrid analysis	6	5.08
5	Secure programming	5	4.24
6	Program transformation	4	3.39
7	Patching	4	3.39
8	SREP	3	2.54
9	Medina-DB	3	2.54
10	SecureUML	3	2.54
11	SREF	2	1.69
12	Microdoft SDL	2	1.69

13	Misuse cases	2	1.69
14	SecureSOA	2	1.69
15	SecureTropo	2	1.69
16	FDAF	2	1.69
17	PbSD	2	1.69
18	SECTET	2	1.69
19	SQUARE	2	1.69
20	CLASP	2	1.69
21	Abuse Frame	1	0.85
22	Apvrille and Pourzandi	1	0.85
23	Van Wyk and McGraw	1	0.85
24	Software Security Assessment Instrument (SSAI)	1	0.85
25	S2D-ProM	1	0.85
26	Abuse case	1	0.85
27	UMLintr	1	0.85
28	AsmLSec	1	0.85
29	ADM-RBAC	1	0.85
30	AMF	1	0.85
31	Georg-AO	1	0.85
32	Gomaa-UML	1	0.85
33	Xu-Petri	1	0.85
34	YU-AC	1	0.85
35	KAOS	1	0.85
36	HTTPUnit	1	0.85
37	Giordano-Access Control	1	0.85
38	Kim-Access Control	1	0.85
39	Mariscal-AC	1	0.85
40	Vela-DB-XML	1	0.85
41	UML state charts	1	0.85
42	SRS-Tool	1	0.85
43	Buyens-LP	1	0.85
44	Hoisl-SOA	1	0.85
45	UML- AC	1	0.85
46	UMLS	1	0.85
47	Agile Security Framework(ASF)	1	0.85
48	ISDF	1	0.85

49	EUC	1	0.85
50	STS-Tool	1	0.85
51	Gupta- Framework	1	0.85
52	AEGIS	1	0.85
53	AsmL	1	0.85
54	FDD	1	0.85
	Total	118	100.00

Table 7 answers our first research question (RQ1) i.e. the security approaches that have been used to incorporate security into software development lifecycles present in the published literature. Also, it answers the second research question (RQ2) regarding the phase in which the identified security approaches are incorporated (i.e. requirement, design, coding). (For more details about these approaches please see Appendix A).

Table 8 depicts the frequency distribution of various security approaches, as cited in the literature. Based on our study, dynamic analysis and static analysis approaches were cited the most with 14.41 % and 11.02 % respectively. These approaches are one of the most proactive security vulnerability detection methods, including buffer overflow and SQL injection whether by execution of the code, as in dynamic analysis, or a building model analysis of the artifact without execution, as in static analysis (for more information please see Appendix A). These percentages may be due to insecure coding guidelines and practices such as SQLAI, or as a result of improper input validation. Furthermore, this is probably due to the fact that many vulnerabilities present in the requirement and design stages will appear again, especially if they are not captured when they are presented. These well-known approaches are used to detect security vulnerabilities such as BOF, XSS and SQLI in the coding phase of SDLC in a proactive

manner, i.e. before the software release. The other approaches range from process, modeling language and notations to frameworks ranked in the list, with percentages between 5.08 % and 0.85%.

5.1.2 Lifecycles Phases Frequency Analysis

Table 9 freq. of studies in security SDLC

Lifecycle Stage	Frequency	%
Coding	52	**41.6**
Design	39	31.2
Requirement	27	21.6
Across	7	**5.6**
Total	125	

Our second research aspect focuses on the phase in which the security approaches are incorporated (i.e. requirement, design, coding) and which stage is covered most by studies published in the literature. Based on our review, the stage in the development lifecycle where a security approach is emphasized varies in different studies. Table 9 shows that a significant number (about 41.6%) of studies in this review considered security checks around the coding stage of development. This is probably because attacks are more likely as a result of improper coding practices, such as SQL injection attacks, buffer overflow (BOF) and cross site scripting (XSS). Alternatively, it may be that vulnerabilities introduced during requirements and design will manifest themselves in code if not detected when they are introduced. However, applying security checks across the entire lifecycle has received less attention and is only considered in 7 out of 125

primary studies. Similarly, there has not been an empirical study (to the best of our knowledge) that assesses whether concentrating security around coding is sufficient or not. However, adding security checks across the entire lifecycle, which also includes the coding stage, will guarantee more assurances than if they are only introduced during the coding stage. Also, we can observe from Table 9 that no results have been identified in the testing phase. This may be due to the fact that, when the software enters the testing phase, all its functionalities are built so we cannot prevent security flaws and bugs and make the software secure, all we can do is find the security issues that exist in the software. Furthermore, in the testing phase the tester needs to put himself in the position of the attacker, which explains how hard the security testing phase is. Similarly, in the maintenance phase, the only thing we can do is find any security issues that exist in the software.

5.1.3 Active researchers Analysis

Table 10 Active researchers Freq. Analysis

Author	#Papers
Jan Jürjens	6
Zulkernine, M.	6
Bashar Nuseibeh	4
Eduardo Fernández-Medina	4
Daniel Mellado	3
Best, B.	2
Charles B. Haley	2
Shahriar, H.	2
Mario Piattini	2

The third research aspect focuses on the most active researchers who contributed to the research topic, which will also help to answer (RQ3). To get an overview of active

researchers in this area, we followed a common metric in software engineering [177]. This metric works by counting the number of papers published by each author. To keep the brevity of the ranking results, we showed the top authors (in order) who have published at least 2 papers in the pool in Figure 11.

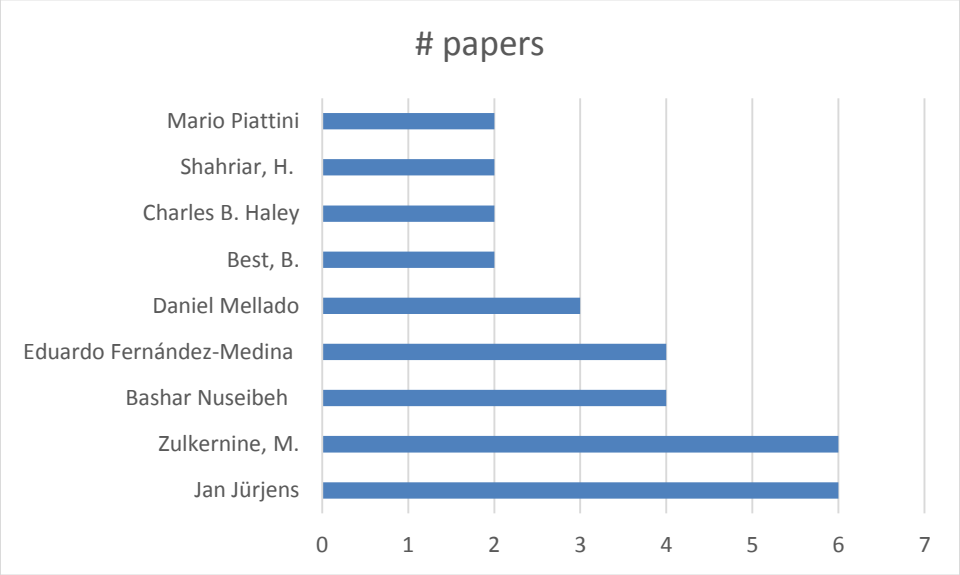

Figure 11 Top 9 researchers in the area

The competition is close, as the second and third ranks tie. The ranking is as follows: Paolo Jan Jürjens and Zulkernine, M. (6 papers each), Bashar Nuseibeh and Eduardo Fernández-Medina (4 papers), Daniel Mellado (3 papers) and Best, B., Charles B. Haley, Shahriar, H., and Mario Piattini each with 2 papers as shown in Table 10. (For more information about the active researchers who contributed to the research topics please see Appendix A.)

5.1.4 Publication Venues and Sources Types Analysis

The fourth research aspect focuses on the publication venues and source types of the published primary studies, which will help to answer (RQ4) (i.e. the most active venue of publication that contributed to software security).

Table 11 Distribution of selected studies over source types.

Publication Channel	frequency	%
Conference	54	**45.76**
journal	33	27.97
Symposium	13	11.02
Book chapter	10	8.47
Lecture note in computer science	5	4.24
Workshop	3	**2.54**
Total	118	100.00

The selected studies were published in six publication types: conferences, journals, symposiums, book chapters, lecture notes in computer science, and workshops. Table 11 shows the distribution of selected studies over publication types. Conferences, journals, symposiums and book chapters are the four main publication types with 45.76% (54 studies), 27.97% (33 studies), 11.02% (13 studies) and 8.47% (10 studies) of the selected studies, respectively. Only 5 studies were published as lecture notes and 3 as workshops, as shown in Table 11.

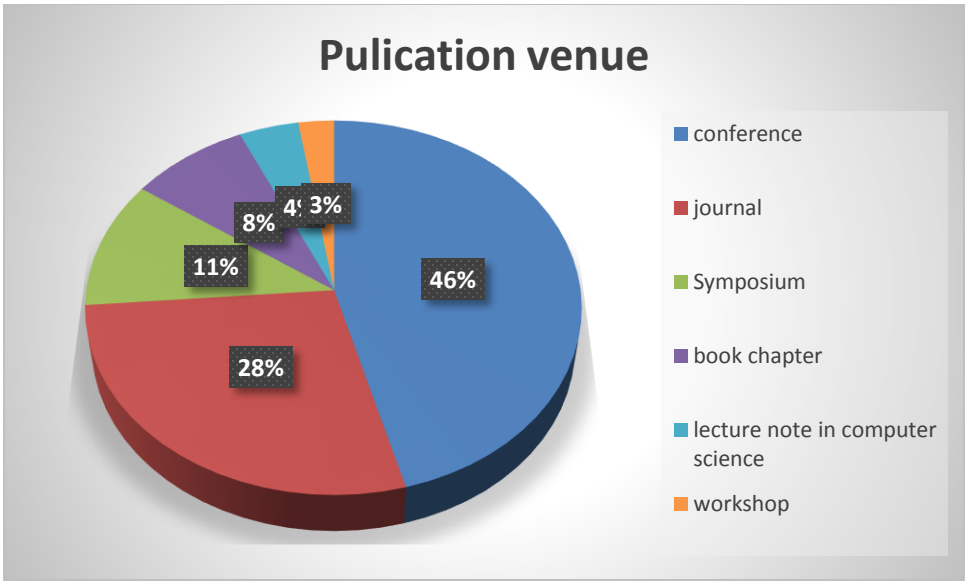

Figure 12 Publications venue distribution

Table 12 Top five Publication venues of identified articles

Publication Venue	Type	No.	%
ICSE 2007. 29th International Conference on Software Engineering, 2007.	Conference	6	5.08
ICSE Workshop on Software Engineering for Secure Systems, 2009. SESS '09.	Conference	5	4.24
Proceedings. Eighth IEEE International Conference on Engineering of Complex Computer Systems, 2002.	Conference	4	3.39
COMPSAC '08. 32nd Annual IEEE International Computer Software and Applications, 2008.	Conference	4	3.39
Software & Systems Modeling	Journal	4	3.39

Table 12 presents the top five publication venues of some the selected studies, their types, the number of studies, and the corresponding proportion against the total number of selected studies. Overall, 82 publication venues are identified that cover different areas of computer science, such as software engineering, security, networking, etc.; which means this study topic has received wide attention in the research community.

One observation that can be made is that there is one leading conference (ICSE), workshop (SESS), and journal (Software & Systems Modeling) respectively as the publication venues for this study topic. Also, we can note that these three venues are in the field of software engineering. This demonstrates the importance of software security research in software engineering and other related fields. **(For more details of these all identified publication venues please see Appendix A)**

5.1.5 Demographic Analysis

Another aspect that needs to be discussed is the demographic analysis (country and continents) of identified publications that have contributed to research topic.

Table 13 Country frequency analysis

Country	Freq.	%
USA	43	**32.58**
Canada	12	9.09
Germany	11	8.33
UK	9	6.82
Spain	9	6.82
Italy	8	6.06
Austria	6	4.55

We ranked the most active countries based on the affiliation of authors who have published software security approaches papers. The rationale for this ranking is to know which researchers from which countries (as a group) focus more on software security (building secure software). If an author had moved between two or more countries, we

attributed each of his/her papers to the explicit affiliation information on top of each paper. If a paper was written by authors from more than one country, we incremented the counters for each of those countries by one.

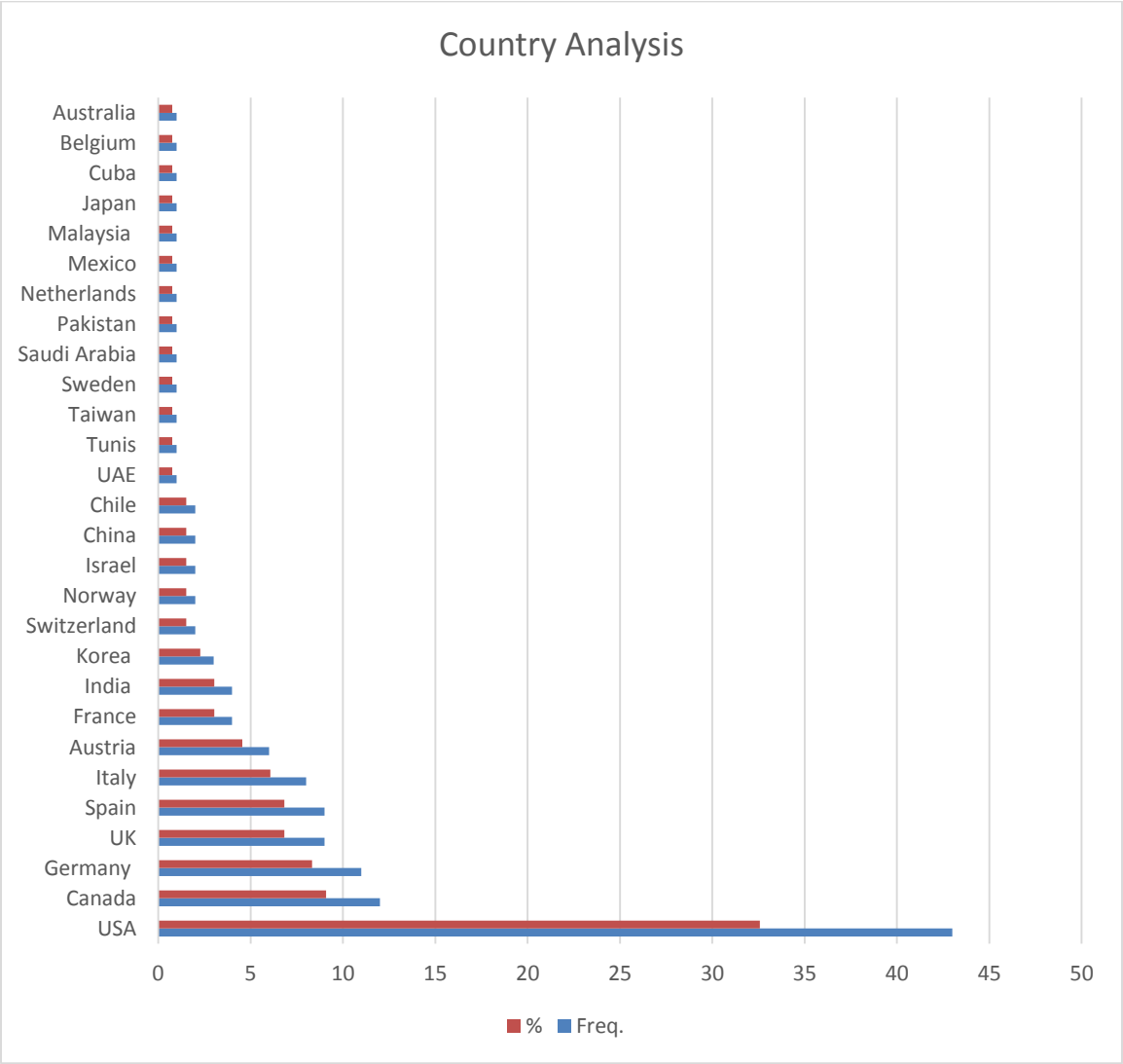

Figure 13 Country contributed to Software Security

It is important to note that Table 13 and Table 13 shows primary studies originating from 28 different countries, because it is vital to examine research from

different social and organizational cultures. The results are shown in Table 13 and Table 13. American researchers have authored or co-authored 32.58 % (43 of 133) of the articles in the pool. Authors from Canada, Germany, UK, Spain, Italy and Austria (with 12, 11, 9, 9, 8 and 6 articles, respectively) stand in the second, third, fourth and fifth ranks. Most of the remaining articles were written by researchers from various countries that contributed between 2 and 4 articles. This indicates that more studies from different countries are needed to account for cultural and social differences that may have an effect on research findings.

Table 14 Continent Analysis

Continent	Freq.	%
Americas	**57**	42.86
Europe	**57**	42.86
Asia	17	12.78
Australia	1	0.75
Africa	1	0.75
Total	133	100.00

For the continents analysis, Table 14 illustrates the distribution of primary study which proposed security approaches over the continents. There are 57 articles are written by authors from America. Similarly, 57 from Europe while small amount of 17 papers are from Asia. Also, the lowest number of papers have been produced in Australia and Africa. This is probably due to that the countries in this regions pay more attention to software security as the software systems are more widely used in daily life.

5.1.6 Study Strategy and institutional Analysis

Table 15 Study Strategy Used

Study Types	Count
Case study	54
Experiment	33
others	31
Total	118

Another aspect that we focused on is the different type of study strategies used to propose security approaches presented in the literature. Table 15 gives a summary of each type of study strategy found in the published literature. We have grouped the papers found through SLR into three study strategies, which are commonly used in empirical software engineering as, shown in Table 15. These study strategies were classified as case studies or experiments. In addition, some articles that could not be clearly classified with the above categories were placed in the 'other' category and the count of these articles was 31. The 'other' category mainly included articles that have developed a new tool, evaluated it and demonstrated it. Furthermore, some of studies developed a new programming language that somehow provides security by its nature. We can observe from the search strategy type, there are no more systematic literature reviews or

systematic mapping studies in this topic, which supports our contribution that few SLRs contribute to the topic and no SLR covers all the SDLC.

Table 16 Institution Analysis

Institution	Country	Freq.(139)	%
Queen's University	Canada	10	**7.19**
University of Castile–La Mancha (UCLM)	Spain	7	5.04
The Open University	UK	5	3.60
Univ. of California	USA	4	2.88
Virginia University	USA	4	2.88

Also, an institutional analysis has been conducted to see an institution's contribution to the research topic. In the 118 papers reviewed, 139 distinct institutions, ranging from universities, research institutes and industrial organizations have been found. Table 16 illustrates popular institutions that have contributed to the topic with more than 3 papers. The Queen's University in Canada obtained the first rank, followed by University of Castile–La Mancha (UCLM) in Spain, with 7.19% and 5.05% respectively. The rest of the institutions contributed between one and three publications on the topic.

5.2 Snowballing and SLR Results for RQ5

This section presents the initial snowballing-based literature survey results. Of the total number of results retrieved by both types of snowballing (i.e. backward and forward snowballing), 47 studies presented the limitations and challenges of incorporating security into different stages of SDLC. In addition to searches of in the databases using the search string, snowballing from the list of references of the identified articles was used as another method in this research, i.e. to identify additional relevant articles through the reference lists of the articles found using search strings and articles identified through a manual search using Google Scholar to answer RQ5. Both backward snowballing from the lists of references and forward snowballing by finding citations to the papers were included in this research. We analyzed each publication and extracted limitations, whether for specific approaches or in general (i.e. limitations of building security from the beginning of software development). Table 17 shows the all the limitations and challenges for existing software security approaches whether for specific approach or in general (i.e. what make building security is harder from the early stage of SDLC). The total primary studies 165 were selected using both SLR and snowballing, these studies were analyzed for achieving our second objectives to identify existing software security approaches limitations in SDLC. Table 17 Shows the limitations and challenges of existing software security approaches in SDLC. Also , table 18 shows the limitations categorizations based on the approaches types.

Table 17 Limitations and Challenges

#	**Limitations and challenges**	**Approach**	**References**
1	Lack of analyzing of security threats and derive of security requirement using the problem frame (Requirement elicitation approaches).	Abuse Frame	[68]
2	Inadequate of security requirement (definition).	SREF	[69] [70]
3	Lack of awareness of security concerns within development team.	Apvrille and Pourzandi	[54]
4	Disconnected between security and software development.	Van Wyk and McGraw	[71]
5	Lack of security activities within normal SDLC.	Microdoft SDL	[50] [72]
6	Lack of security activities within normal SDLC.	Software Security Assessment Instrument (SSAI)	[73]
7	Lack of security engineering expertise between engineer and developer to check the level of security.	S2D-ProM	[74]
8	• Use case is poor at supporting security requirements. • Limited support of security threats and requirement using use case.	Misuse cases	[51] [75]
9	Lack of expertise to derive security requirements between development team.	Abuse case	[59]
10	Lack of specifying intrusion detection in UML notations.	UMLintr	[76]
11	Lack of specifying attack scenarios specification in modeling language.	AsmLSec	[77]
12	Unsupported of specifying access control security property in designing web applications.	ADM-RBAC	[78]
13	Lack of supporting security concerns in SDLC (authorization access control).	AMF	[79]

14	The need of Integrating security concerns in design level.	Georg-AO	[80]
15	Absence of separating between security concerns and business requirement.	Gomaa-UML	[81]
16	Lack of presenting and analyzing security policies with in the participant of SOA.	SecureSOA	[82] [83]
17	Lack of security information and modeling security properties in UML diagrams.	UMLsec	[84] [85] [53] [57] [55] [86]
18	Lack of modeling and verifying of secure software and threat behavior.	Xu-Petri	[87]
19	Lack of presenting and analyzing of security properties using UML.	YU-AC	[88]
20	Lack of considering and specifying security in KAOS (goal oriented approach that used in the requirement phases).	KAOS	[52]
21	Lack of security concerns specification in Agile development.	HTTPUnit	[89]
22	• Limited to some security vulnerabilities such as Buffer overflow, cross site scripting and SQL injection, while other vulnerabilities such as cross site request forgery and format string bug does not addressed by this approaches.	Dynamic analysis	[90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [101] [102] [103] [104] [105] [106]
23	• Suffer from false positives and false negatives. • Only support high level language such as PHP , Java and C. • Limited to some type of vulnerabilities such as Buffer overflow, format string bug and SQL injection, while other vulnerabilities such as cross	Static analysis	[107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] [118] [119]

	site scripting and cross site request forgery do not supported.		
24	• None of this approaches addressed cross site request forgery (CSRF). • Limited to analyze code written using scripting language such as PHP and JSP, and in procedural language (C). • Limited to address few types of vulnerabilities such as SQLI and XSS.	Hybrid analysis	[120] [121] [122] [123] [124] [125]
25	• Limited to few types of vulnerabilities such as buffer overflow, SQL injection and cross site scripting. • Only two programming language supported by this approach (C and JAVA).	Secure programming	[126] [127] [128] [129] [130]
26	• Limited to some types of vulnerabilities (BOF and XSS). • Deal with source code written using C and Java Script.	Program transformation	[131] [132] [133] [134]
27	• Limited to some vulnerabilities such as BOF, SQL and XSS. • Limited to code written using C programming language.	Patching	[135] [136] [137] [138]
28	Lack of standard-based (CC) process that deal of security at the early stages of software development.	SREP	[139] [140] [141]
29	Security does not supported by Tropo a normal requirement elicitation approach.	SecureTropo	[142] [143]
30	Lack of presenting and analyzing of security property in UML class diagram.	FDAF	[144] [145]
31	Lack of modeling access control security policy in	Giordano-Access Control	[146]

	the early stage of SDLC.		
32	Need for presenting and analyzing role based access control security policy in the design phase.	Kim-Access Control	[147]
33	Neglecting visualizing the access control security policy in the design phase of SDLC.	Mariscal-AC	[148]
34	The need of eliciting and development security concerns in the whole data warehouse development lifecycles.	Medina-DB	[149] [150] [151]
35	Lack of incorporating security aspect related to database access authorization with development process.	PbSD	[152] [153]
36	Neglecting of security (confidentiality security policy) during the development XML data warehouse.	Vela-DB-XML	[154]
37	• The need of modeling designs along with their security policy. • The need of making transition from designs and policies to secure systems. • Neglecting of engineering security in overall development software system.	SecureUML	[56] [155] [156]
38	The need of presenting and modeling intrusion detection in the early stage using software specifications language.	UML state charts	[157]
39	The need of guidance as a security requirement development process.	SRS-Tool	[158]
40	Lack of consistent framework and methodologies for modeling security concerns within SOA participants.	SECTET	[159] [160]
41	Neglecting of security design principle (Least Privilege) by software architects.	Buyens-LP	[161]
42	Demands for a through integration of security	Hoisl-SOA	[162]

	features (confidentiality and integrity) in the development process of service-oriented systems.		
43	The need of specifying the security policies (access control) in the normal software engineering models.	UML- AC	[163]
44	• Neglecting confidentiality issue during SDLC. • The need of UML notations that support this issue.	UMLS	[164]
45	Lack of security practice in Agile development.	Agile Security Framework(ASF)	[165]
46	Lack of knowledge and skills needed of secure software system.	ISDF	[166]
47	• Lack of security knowledge and skill to analyzing and eliciting security requirement among software engineer. • Neglecting security requirement in the early stages.	EUC	[167]
48	Lack of modeling the secure social-technical system.	STS-Tool	[168]
49	• Absence of presenting security concerns in software architectural level. • Lack of supporting secure design decision.	Gupta-Framework	[169]
50	Lack of systematic security requirement engineering in the early stage of software development.	SQUARE	[170] [171]
51	Lack of systematic security requirement engineering in the early stage of software development.	CLASP	[172] [173]
52	The need of approach that make a Trade-off between secure and usable system.	AEGIS	[174]
53	Lack of writing security requirement (Intrusion detections) using normal software specification language.	AsmL	[175]

54	Absence of integrating security concerns in Agile development.	FDD	[176]
55	• Knowledge of stakeholders, programmers and testers. • Disregard of security which results from deficient knowledge of stakeholders.	General	[178][179][167][180][181][182] [183][184] [185][186]
56	Need for proofing- lack of empirical studies.	General	[187][188][189][190][191][192][193][184][194]
57	The need of Security Experts Involvement.	General	[195][196][197][198][199][167] [200][190]
58	Limited to some security concerns.	UMLintro, SecureUML	[167][201][61] [184][202][203]
59	Suffer from false positive and false negative.	Static analysis	[204][205][194][206][207][208][209]
60	The need of supporting Secure - A Social-Technical.	SecureTropos	[210] [211] [212][213][214]
61	Scalability. (This limitation specific to the UML-based approaches, the nature of their analysis lead to limitations in the complexity of the interactions they can support making them unfit for modeling large systems.)	UML-based approaches (Misuse case, UMLsec and SecureUML).	[215][216][217][214]
62	Neglecting inside threats. The notion of misuse cases cannot explain why a misuser attacks the system, and the impact of a security use case and a misuse case on other use cases	Misuse cases	[218][184][219]
63	Does not cover all the coding vulnerabilities.	Static analysis, dynamic analysis, secure programming, patching.	[205][185] [209]

64	Limited to a few Programming language.	Vulnerabilities-mitigation approaches	[205][220] [209]
65	Organizational Impact and business processes. This limitation is caused by too much organizational focus on time-to-market and the usability of the system. Also, software development teams are constantly under severe pressure and deadlines to meet delivery dates and customer commitments.	General	[195][221]
66	No more comprehensive approach that cover all stages.	General	[201][202]
67	Learnability and understandability. The technique is learnable in a definite and acceptable time period. Also, there should be clear steps and activities for the technique.	Misuse case	[217][216] [219]
68	Traceability. Traceability means being able to keep track of the history of how models are generated throughout the software lifecycle, and how they relate to each other. It helps to trace design flaws back to a model when a counterexample is detected during the verification of less abstract model, or errors are found during the testing of the produced system's infrastructure	SecureUML , UMLsec , SECTET	[222]
69	Used in the industry. It has been reported that a small number of proaches (processes oriented) have been used in the lustry, such as Microsoft SDL and CLASP.	General	[8]

70	Others	UML-based approaches	[214] [184]

Table 18 limitations and challenges categorization

Group	Approach	Limitations and challenges
Reverse approach ***(****Consider security requirements in a reverse way, e.g. identifying problems or attacks that may subvert the security of software systems****).***	Abuse frame	Lack of analyzing of security threats and derive of security requirement using the problem frame (Requirement elicitation approaches).
	Misuse cases	• Use case is poor at supporting security requirements. • Limited support of security threats and requirement using use case. • Neglecting inside threats. • Learnability and understandability. • The lack of a precise set of guidelines for their definition, which renders them unsuitable for certain kinds of threats, especially when a large number of critical assets are involved. • This method also fails to provide guidance on when and how identified security issues can be tackled and how the produced security requirements can be linked to the rest of the development process.
	Abuse case	Lack of expertise to derive security requirements between development team.
	Essential Use case	• Lack of security knowledge and skill to analyzing and eliciting security requirement among software engineer. • Neglecting security requirement in the early stages.
Process Oriented ***(****Proper steps, procedures activities to guide the participants.)*	SREF	Inadequate of security requirement (definition).
	Apvrille and Pourzandi	Lack of awareness of security concerns within development team.
	Van Wyk and McGraw	Disconnected between security and software development.

	Microdoft SDL	Lack of security activities within normal SDLC.
	Software Security Assessment Instrument (SSAI)	Lack of security activities within normal SDLC.
	S2D-ProM	Lack of security engineering expertise between engineer and developer to check the level of security.
	SREP	Lack of standard-based (CC) process that deal of security at the early stages of software development.
	ISDF	Lack of knowledge and skills needed of secure software system.
	SQUARE	Lack of systematic security requirement engineering in the early stage of software development.
	CLASP	Lack of systematic security requirement engineering in the early stage of software development.
	AEGIS	• The need of approach that make a Trade-off between secure and usable system. • Limited to some security concerns.
UML-based approaches ***(*** *Approaches that make use the Unified Modeling Language notations)* *(This group of approaches cannot serve projects of different sizes(scalability limitations)*	UMLintr	Lack of specifying intrusion detection in UML notations.
	Georg-AO	The need of Integrating security concerns in design level.
	Gomaa-UML	Absence of separating between security concerns and business requirement.
	YU-AC	Lack of presenting and analyzing of security properties using UML.
	FDAF	Lack of presenting and analyzing of security property in UML class diagram.
	Kim-Access Control	Need for presenting and analyzing role based access control security policy in the design phase.

	Mariscal-AC	Neglecting visualizing the access control security policy in the design phase of SDLC.
	Medina-DB	The need of eliciting and development security concerns in the whole data warehouse development lifecycles.
	PbSD	Lack of incorporating security aspect related to database access authorization with development process.
	UMLsec	• Lack of security information and modeling security properties in UML diagrams. • The resulting models do not express attackers' behavior, and the threat description is limited, using the notion of Delete, Read and Insert stereotypes to change a state of the subsystem.
	SecureUML	• The need of modeling designs along with their security policy. • The need of making transition from designs and policies to secure systems. • Neglecting of engineering security in overall development software system. • Limited to some security concerns.
	UML state charts	The need of presenting and modeling intrusion detection in the early stage using software specifications language.
	Hoisl-SOA	Demands for a through integration of security features (confidentiality and integrity) in the development process of service-oriented systems.
	UML- AC	The need of specifying the security policies (access control) in the normal software engineering models.
	UMLS	• Neglecting confidentiality issue during SDLC. • The need of UML notations that support this issue.

Notations ***(****Specify and present security specification (security properties, attack specification, security requirement) using new proposed notations.)*	ADM-RBAC	Unsupported of specifying access control security property in designing web applications.
	AMF	Lack of supporting security concerns in SDLC (authorization access control).
	Xu-Petri	Lack of modeling and verifying of secure software and threat behavior.
	Giordano-Access Control	Lack of modeling access control security policy in the early stage of SDLC.
	Buyens-LP	Neglecting of security design principle (Least Privilege) by software architects.
	SECTET	Lack of consistent framework and methodologies for modeling security concerns within SOA participants.
	AsmLSec	Lack of specifying attack scenarios specification in modeling language.
	AsmL	Lack of writing security requirement (Intrusion detections) using normal software specification language.
	SecureSOA	Lack of presenting and analyzing security policies with in the participant of SOA.
Vulnerabilities-Mitigation approaches ***(****Dealing with common security vulnerabilities in the coding phase such as BOF, XSS, SQLI ...)* *(Limited to a few Programming language.)*	Static analysis	• Suffer from false positives and false negatives. • Only support high level language such as PHP, Java and C. • Limited to some type of vulnerabilities such as Buffer overflow, format string bug and SQL injection, while other vulnerabilities such as cross site scripting and cross site request forgery do not supported.
	Dynamic analysis	Limited to some security vulnerabilities such as Buffer overflow, cross site scripting and SQL injection, while other vulnerabilities such as cross site request forgery and format string bug does not addressed by this approaches.

	Hybrid analysis	• None of this approaches addressed cross site request forgery (CSRF). • Limited to analyze code written using scripting language such as PHP and JSP, and in procedural language (C). • Limited to address few types of vulnerabilities such as SQLI and XSS.
	Secure programming	• Limited to few types of vulnerabilities such as buffer overflow, SQL injection and cross site scripting. • Only two programming language supported by this approach (C and JAVA).
	Program transformation	• Limited to some types of vulnerabilities (BOF and XSS). • Deal with source code written using C and Java Script.
	Patching	• Limited to some vulnerabilities such as BOF, SQL and XSS. • Limited to code written using C programming language.
Goal-oriented Approaches *Extended for goal oriented modeling approaches that focuses on describing both organizational environment of a system and a system itself. Also, it extended for specifying the anti-goal and constrains of the systems.*	KAOS	Lack of considering and specifying security in KAOS (goal oriented approach that used in the requirement phases).
	SecureTropo	• Security does not supported by Tropo a normal requirement elicitation approach. • The need of supporting Secure - A Social-Technical.

Others *(specific to certain technology , development methodology or general limitations)*	HTTPUnit	Lack of security concerns specification in Agile development.
	Vela-DB-XML	Neglecting of security (confidentiality security policy) during the development XML data warehouse.
	SRS-Tool	The need of guidance as a security requirement development process.
	Agile Security Framework(ASF)	Lack of security practice in Agile development.
	STS-Tool	Lack of modeling the secure social-technical system.
	Gupta-Framework	• Absence of presenting security concerns in software architectural level. • Lack of supporting secure design decision.
	FDD	Absence of integrating security concerns in Agile development.
	General (general limitation for incorporating security in SDLC)	• Knowledge of stakeholders, programmers and testers. • Disregard of security which results from deficient knowledge of stakeholders.
	General	Need for proofing- lack of empirical studies.
	General	The need of Security Experts Involvement.
	General	Organizational Impact and business processes.
	General	No more comprehensive approach that cover all stages.
	General	Used in the industry- only CLASP and SDL.

CHAPTER 6

CONCLUSION

Traditionally, software security is only considered in the later stages of software development with the incorporation of security concerns as an afterthought. As a consequence, the risk of introducing new security vulnerabilities into various stages of software development lifecycles increases. Research evidence has proven that approaches to address security-related concerns are insufficient and could likely cause costly reworks in addition to all the intangible consequences caused by a security breach. To avoid these costly mistakes, security concerns need to be addressed from the beginning of software development lifecycles all the way through to deployment and maintenance. Several approaches have been proposed in the literature for incorporating security into the SDLC from the requirements gathering phase until the maintenance and deployment, along with recommended tools to support a security-centric software development lifecycle. Despite the importance of these approaches, only a small amount of research has been carried out to investigate the approaches and their limitations in a systematic manner.

With this focus, we were interested in exploring software security approaches and their limitations in the software development lifecycle. This research aimed at exploring software security approaches and their limitations in software development lifecycle by tackling five research questions. The main results are as follows:

RQ1: 118 articles were selected using SLR that met our inclusion criteria and quality assessments. We analyzed each publication and extracted about 54 relevant approaches

for incorporating security into different phases of SDLC. Based on our study, dynamic analysis and static analysis approaches were cited the most with 14.41 % and 11.02 % respectively. These approaches have been categorized (for better understanding) mainly into seven main categories (Reverse approach, Process Oriented, UML-based approaches, Notations, Vulnerabilities-Mitigation approaches, Goal-oriented Approaches and Others). (More information see section 5.1 – 5.1.1)

RQ2: Based on our research, the phase in the software development lifecycle where a security approach is emphasized varies in different studies. The result shows that a significant number of studies in this review considered security checks around the coding phase of development. However, applying security checks across the entire lifecycle has received less attention. (More information see section 5.1.2)

RQ3: The third research aspect focused on the most active researchers who contributed to the research topic. To get an overview of active researchers in this area, we followed a common metric in software engineering [177]. This metric works by counting the number of papers published by each author. (More information see section 5.1.3)

RQ4: With respect to the publication venue and study type were the selected studies published, the selected studies were published in six publication types: conferences, journals, symposiums, book chapters, lecture notes in computer science, and workshops. Also, Overall, 82 publication venues were identified that cover different areas of computer science, such as software engineering, security, networking, etc. (More information see section 5.1.4)

RQ5: The total primary studies 165 were selected using both SLR and snowballing, these studies were analyzed for achieving our second objectives to identify existing software security approaches in SDLC. We analyzed these studies and retrieved various limitations and challenges whether for identified existing software security approaches or general challenges and limitations. (More information see section 5.2).

Two research methodologies are used in this thesis: SLR and Snowballing. SLR is intended to provide a comprehensive scanning of all the articles targeting the software security approaches in software development lifecycles. Its main purpose is to explore software security approaches proposed in the literature and to examine which stages the identified approaches emphasized. Fifty-four security approaches are identified for providing security checks in various phases of software development, and significant analysis has been conducted, including demographic analysis to reveal any hidden patterns. Furthermore, the publication venues and active researchers who contributed to the topic are identified. Snowballing was used as another method in this research to identify possible limitations for incorporating security into software development lifecycles. Various limitations have been identified, including knowledge of the software development teams and the need for security experts to be involved in most of the security approaches, especially in the requirements phase. Also, the limitations for each software security approaches have been identified as described in the literature.

6.1 Contribution

Few research articles discuss the software security approaches in software development lifecycles. The articles that exists were produced in the last few years, and some carried out reviews and comparison studies on the issue. Most of these reviews focused only on secure software development at the requirements engineering phase of the SDLC, and others concentrated only on investigating security practices for special software development methodologies, such as agile or XP. However, based on our research, none of them performed a review focused on software security approaches that cover all stages of software development lifecycles in a systematic manner using systematic literature review, and none of those reviews documented the systematic processes for selecting the primary studies. Also, none of the existing articles explore the limitations of incorporating security into software development lifecycles. Our work is of high value and can serve as a reference for understanding the various software security approaches into software development lifecycles and their limitations. Our research can be considered a first stone that assists software development organizations in better understanding the existing software security approaches used in the software development lifecycle and their limitations. It can also provide other researchers with a firm basis on which to develop new software security approaches.

6.2 Validity

The results of this research are based on systematic literature review and snowballing. Despite our extreme care to provide accurate and valid data, a few points must be considered when adopting our results. We tried to design our search string to

cover all the software development lifecycles from the requirement phase through to deployment and maintenance, but a threat to validity stems from the fact we do not include all articles that proposed software security approaches which may affect the completeness of the study search. To mitigate this threat, we used snowballing as another research methods to identify the limitations of exiting software security approaches in software development lifecycles.

6.3 Lesson Learned

This thesis is the result of a full year and a half of work and effort. The experience is indispensable, and the obtained knowledge is of great value. Conducting a systematic literature review is a very demanding task. It requires reading an enormous number of papers quickly and then determining the right ones to evaluate based on quality criteria. I have learned how to use the advanced settings on various academic database search engines. I have also learned how to use synonyms of terms to retrieve additional relevant materials. Synthesizing the results and correlating the information with study types, publication venues, and active researchers strengthen my skills as a researcher. I am able to see issues from various perspectives and connect them in order to detect any hidden patterns. Additionally, I have learned how to use snowballing as a second method in my research to identify more results related to specific research questions.

6.4 Future Work

Our work stems from the fact that there are few articles and reviews that explore the software security approaches in a systematic manner, and none of them address the limitations of incorporating security into software development lifecycles. As a part of

future work, secure software development is an ongoing research area, and we can enhance any of software security approaches that have been identified in this research, or we can address one of the identified limitations for incorporating the security into software development lifecycles. Also, Empirical research in the real industrial environment to explore the limitations of software security approaches in SDLC will be conducted.

APPENDIX

Software Security Approaches Details

	Approach	Description
1	Dynamic Analysis (Testing Security Vulnerabilities)	Dynamic analysis is one of the most proactive approaches used for mitigating code security vulnerabilities, such as buffer overflow and SQL injection attacks, before the software release. The tested program implementation is checked with specific input during its execution and then both of the computed and expected outputs are compared. Then the mismatches between the inputs and expected outputs are checked. If there are mismatches between them, the implementation does not satisfy the desired security objectives (i.e. requirements) of the particular input. Three major processes need to be performed in this approach: 1. Identify the requirements and coverage: Based on the functional requirements, it is necessary to identify the security requirements. In this case, this is the security vulnerabilities generated by the software implementations, invalidated inputs, et cetera. The security breaches, such as buffer overflows and SQL injection attacks, are defined in advance.

		2. Generate test case: In this step, the program artifacts such as source code and executable code are used for generating test case. Then the presence or absence of an attack is determined by the state of the program. Various test case generations have been used, including fault injection and mutant analysis. 3. Test case execution: In this step, the generated test case is run against the implementation to determine the presence or absence of the security vulnerabilities defined based on the attack symptoms. Various test case generations are used by these approaches, including fault injection and mutant analysis. In fault injection, the input data and variable are corrupted, and the program executes with the corrupt data. Based on that, the expected responses confirm the presence or absence of the vulnerabilities. Also, in fault injection, the user can modify the state of the program (i.e. variables or sensitive locations in the code, such as functions that control other locations) to check whether the program can handle the vulnerabilities. For example, the user can change the structure of the HTML file by replacing one tag with another and check for the presence or absence of the vulnerabilities. Similarly, mutant analysis test generation is a modified implementation of the fault injection type. This mutant describes a rule for injection a fault.

2	Static Analysis	Static analysis is one of the most proactive approaches used for detecting security vulnerabilities, such as buffer overflow, cross-site scripting, and SQL injection attacks, in the program code before the software release. Static analysis was developed for compiler optimization issues and then used for detecting security vulnerabilities due to widespread security issues. Static analysis works by scanning one or more source files and creating a representation of the scanned source to analyze it. The input program code is examined, specific rules, called inferences, are applied to that code, and then a list of vulnerabilities that exist in the code is derived. Inference is the core part of this approach, in which the code is scanned. Various types of inference rules of static analysis have been proposed, including tainted data flow and annotation inference. In tainted data flow inference, the approach marks the input variable as tainted, and then their propagations are tracked. Based on that, warring is generated if the tainted input participates in sensitive operations. Furthermore, in annotation-based inferences, the approach annotates the code with interested properties in term of post-condition and pre-condition and then checks whether the input can be used safely, based on the annotation previously created. All these tools follow the same pattern when applied to a piece of

		source code: 1. Transforming the code to be analyzed into a program model, which is a set of data structures that represent the code. 2. Analyzing the model using different rules and/or properties. 3. Showing the results to the analyzer.
٣	UMLsec	The UMLsec approach is an extension of unified modeling language used for securing the development system. It specifies security requirements using stereotypes, tags, and constraints. UMLsec uses stereotypes as a label in the UML diagram for presenting constraints that need to be achieve by the model and tags for specifying simple properties of model elements. Twenty-one stereotypes have been defined for presenting security requirements. These stereotypes can be associated with various diagrams to represent security requirements and design specifications in the UML model, such as deploy diagram , use case diagram, sequence diagram, class diagram, activity diagram, and state chart diagram.
٤	Hybrid Analysis	Due to the pros and cons of static analysis and dynamic analysis, a huge number of test cases in both dynamic and static analysis produce false positive and negative results. Hybrid analysis is a combination of the two complementary approaches. To minimize the disadvantages of

		static and dynamic analysis, this hybrid type only examines the suspected vulnerable code by identifying the location of the program code that needs to be analyzed and check it against actual exploitations of vulnerabilities. Using dynamic analysis, the actual exploitation of vulnerable code is verified with input test cases.
٥	Secure Programming	Lack of understanding of the programming language, such as data types or libraries, and programmers neglecting possible vulnerabilities are considered dire practices that contribute to the writing of vulnerable code. The secure programming approach supports writing vulnerability-free code in such forms as APIs, language, safe libraries, and filters.
٦	Program Transformation	The program transformation approach is one of the most popular techniques used in transforming vulnerable source code into vulnerability-free source code. This approach is categorized into source-to-source translation and code rewriting. In source-to-source translation, the enhanced source code is generated automatically from the vulnerable source code in the same language. In the case of code rewriting, the output of the code is rewritten in another processor. For example, the user could rewrite the vulnerable JavaScript code to stop cross-site scripting (XSS) attacks.

٧	Patching	In the patching approach, the processes take place after the attack occurs. Also, in this approach, the vulnerable code is identified, and the program is modified to remove the vulnerabilities. Two types of patching have been proposed, source code patching and environment patching. In source code patching, the source code is analyzed to identify the vulnerable statements that need to be fixed. This approach helps in rewriting the SQL statements in such a way the query structure does not change due to malicious code. Also, this approach helps avoid the BOF by replacing the unsafe library with a safe equivalent. In environment patching, the process is completed without stopping the execution. For example, a BOF attack can be avoided by redirecting vulnerable functions such as *strcpy* with equivalent invulnerable functions such as *strncpy*.
٨	SREP (Security Requirement Engineering Process)	SREP is a standard-based process that supports security requirements during the early stages of software development lifecycles in a systematic manner. This approach is based on Common Criteria as an international standard to achieve comparability between the results of independent security evaluations of IT products. Also, developing the security requirement based on identified a group of activities and roles. These activities define the security, assets, and threats and elicit the security requirements. Additionally, it uses other approaches, such as UMLsec, as assistants to do the proposed activities. Also, this process

		uses UML cases for modeling security objectives, misuse cases for eliciting threats, and groups of templates for ranking threats, attacks, and risk. Nine activities are involved in this process: 1. Agree on definitions 2. Identify vulnerable and/or critical assets 3. Identify security objectives and dependencies 4. Identify threats and develop artifacts 5. Risk assessment 6. Elicit security requirements 7. Categorize and prioritize requirements 8. Requirements inspection 9. Repository improvement
٩	Medina-DB	Secure engineering processes for data warehouses are proposed. The Medina-DB approach models access controls and logging policies for databases in UML class diagrams. Tagged values specify the security level and roles related to classes. OCL constraints express more complex rules. Based on such, UML diagram platform-specific implementations for different databases can be generated. In this

		approach, the transformation between CIM, PIM, and PSM is guaranteed.
١٠	SecureUML	The SecureUML approach is an extension of UML that specifies RBAC policies, which are considered security requirements. Also, in this approach, the class diagram is annotated with related access control information with defined vocabulary. Furthermore, OCL is used for specifying the constraints for permissions, resources, and actions. Moreover, in this approach, code and PSM can be generated automatically.
١١	SREF (Security Requirement Engineering Framework)	The SREF approach is an iterative process that consists of four processes for integrating security requirements with requirement engineering. The five processes are as follows: 1. Capture the functional requirements 2. Find the security goals 3. Identify the assets which are anything that has a value in the organizations 4. Identify security requirements 5. Build satisfaction arguments to help verify the satisfaction of the security requirement
١٢	Microsoft-SDL	The Microsoft-SDL approach is a process proposed by Microsoft for

		incorporating security into SDLC with various security activities during software development. In the requirement specification phase, this approach suggests that the security needs and features are identified using user demand. Also, various activities have been suggested by this process in the design phase, including identifying the components that are critical to security, caring about the application of least privilege principles, minimizing the attack surfaces, identifying entry and access points, modeling the threats and risk analysis on components, mitigating threats by identifying the security requirements, and other activities for secure design. Moreover, various secure coding standards have been recommended by this process in the implementation phase and confirmed by using static analysis and reviewing the code at the end of the phase. Finally, code reviews and security testing should be performed on the complete software to verify it for the final step.
١٣	Misuse Cases	The misuse case approach is an extension of the use case approach to present the unwanted behavior developed in the system. Misuse cases are initiated by misusers, and use cases are intimated by normal users to achieve functionality. Use cases are used to present the requirements, but misuse cases present the security threats.
١٤	SecureSOA	SecureSOA is a security design language used to define the security requirements of service-oriented architectures, which are behaviors of

		the interaction between the participants in the SOA. The concrete syntax of the SecureSOA security design language is stereotype UML class.
١٥	Secure Tropos	The Secure Tropos approach is an extension of a development methodology called Tropos. In this approach, various notions have been used for actor, goal, soft goal, task, resource, security constraint, secure task, and secure resource. An actor can achieve the objective to accomplish a goal and depend on another actor. This is called the dependee and dependent relationship. Security requirements and design specifications can be represented using secure Tropos notation as an interaction between different actors to accomplish certain goals. Also, this approach uses the I* notation language to specify the dependences between different actors.
١٦	FDAF (Formal Design and Analysis Framework)	FDAF uses aspects such as access control to add security properties to UML class diagrams. The available aspects can be stored in an aspect library and woven into the design at hand when needed. Furthermore, FDAF aims to allow the translation of extended UML models to formal languages to facilitate formal analysis. Note that FDAF is not limited to security and also considers performance. Building a role-based access control to the software architecture of an online banking system is a way to illustrate this approach. This approach is used to integrate

		access control to the architecture design.
١٧	PbSD (Pattern-based Method for Secure Development)	The PbSD utilizes the security pattern to enforce the security in the system design. PbSD models templates of common security patterns such as RBAC using UML augmented with template OCL constraints. It helps the designer, particularly the database designer, create a database that complies with the organizational policies relating to the authorizations in the early stages of software development. These patterns are initiated into application models, such as UML class diagrams. The approach has been validated in a controlled experiment with students as participants. In the experiment, PbSD is compared with plain SQL and Oracle's VPD with respect to modeling access control policies.
١٨	SECTET	The SECTET approach models security requirements for service-oriented architectures in UML activity and class diagrams. Both diagrams are extended with several stereotypes. Furthermore, more complex rules are specified with SECTET-PL, an OCL-like policy language. It also automatically generates code for a variety of target platform technologies.
١٩	SQUARE (Security Quality Requirements Engineering)	SQUARE is a comprehensive methodology for integrating security from the early stages of the software development process. It consists of nine steps:

		1. Agree on definitions 2. Identify security goals 3. Develop supporting artifacts 4. Perform risk assessment 5. Select elicitation techniques 6. Elicit security requirements 7. Categorize requirements 8. Prioritize requirements 9. Inspect requirements
٢٠	CLASP	The CLASP approach is a group of secure software developments that perform based on the roles during development. It suggests that security experts should be involved from the beginning of development. According to this approach, it has been suggested to use risk analysis and threat modeling during the requirement specification and design phases. Also, security information has been annotated with class diagrams. Security code reviews and static analysis are recommended in the security assurance phase. Furthermore, a list of common vulnerabilities and how to mitigate them is provided during the development.

		Three activities have been proposed by this process to achieve security requirements: 1. Identify the resources, assets, and roles with the owners and asset users 2. Categorize the resources and assets in classes based on security requirements 3. Identify possible interactions between the resources and assets Various security services are identified based on the interaction between the assets and resources, including accountability, authorization, availability, authentication, confidentiality, and integrity.
٢١	Abuse Frame	Abuse frame is based on problem frame to define anti-requirements (i.e. requirements for malicious users) and abuse frame to analyze security threats. Problem frame helps analyze problems to be solved where interaction between the software and domains in the system context is described. Problem frames are useful in requirement engineering because they help with decomposing the system context into simpler sub-problems mapped to well-known problem classes. This approach is an extension of problem frame, so it utilizes the problem to define the system context. The abuse frame is used for identifying the malicious users within the system context by finding the problem and the sub-problems by utilizing the problem frame. The

		security need is described as constraints on the identified functionalities. The abuse frame diagram is constructed for identifying the threats, and the security need is negated to identify the anti-requirement and present them in the abuse frame diagram. Security vulnerabilities, such as "Limit the number of tries for entering passwords," are identified, and the security requirements are addressed.
٢٢	Apvrille and Pourzandi	This approach is a process proposed by Aprville and Pourzandi for secure software development lifecycles based on their experiences. In the requirement phase, the approach used to identify the high level security objectives such as confidentiality and availability, for software system to be. Also, for the low level, this process uses threat modeling for building the security requirements. The prioritization of these security requirements is based on the results of the risk analysis. For the other phases, this process suggests using UMLsec for presenting the design decision. Also, for the implementation phase, this process advises using a suitable programming language that achieves the security purposes and some security practices for mitigating security vulnerabilities, such as FSB and BOF. Finally, for the assurance phase, various tools have been suggested for use in this phase, including code reviews and static analysis scanner tools.
٢٣	Van Wyk and McGraw	The Van Wyk and McGraw approach suggests security practices that could be applied through the software development lifecycles based on

		the long successful experience of the industry. This approach has been used in organizations with successful results. One of the methods used in the requirement phase is an abuse case. The authors have a lot of experience in this field, and they work as consultants in one of the leading companies in software security.
٢٤	SSAI	The SSAI approach is a group of activities that help in developing secure software using suitable resources and tools. First, this process provides an online database that contains information about vulnerabilities and exploitation and mitigation processes. Another resource provided by this process is a security checklist that helps with developing software in a secure manner. Also, the author explains the details about how to build the checklist and the appropriate items that can be involved. Moreover, this process categorizes a group of security static analysis tools. Finally, the testing tools use security property as the first step to test the software.
٢٥	S2D-ProM (Secure Software Development Process Model)	Secure software development activities have been proposed by the S2D-ProM process. It suggests using risk analysis during various stages of SDLC, such as requirement specification, design, and implementation, and the identified risk can be mitigated using security mechanisms. Based on this process, risk analysis can be done in various phases of SDLC. For example, the user can use personnel experience in the requirement phase and design review in the design

		phase. Also, this process proposes flexible options when proceeding from one stage to another. For instance, source code can be developed from designs based on the secure coding rules or personnel experiences.
٢٦	Abuse Case	The abuse case approach uses UML use case diagrams to present unwanted behaviors in a piece of software. In this approach, the abuse case model is developed and used to present the harmful interactions between normal users (actors) and the abuse cases.
٢٧	UMLintr	The UMLintr approach is UML extension. Different stereotypes and tags are used for attacks specification using various diagrams, such as use case diagrams, state chart diagrams, package diagrams, and class diagrams. Different types of attacks are presented in this approach, including remote to user or denial services, and stereotype packages are used to present each type. For each class, there are three types of stereotypes and 12 stereotypes for each use case diagram. Stereotypes also have tags for classes.
٢٨	AsmLSec (Abstract State Machine Language)	AsmLSec is AsmL extension for attack specifications scenarios. It helps identify how the system under development copes with potential attacks by using knowledge about past attacks observed on similar applications. With this approach, the attacks are represented using transitions, states, and events. There is a source and a destination state

		for each transition. To fire the transition, to achieve the transition from one state to the other, a set of conditions need to be met. Moreover, this approach has the potential of presenting the attack scenarios in AsmL using appropriate compilers, and these attack scenarios can be translated as inputs using intrusion detection systems.
٢٩	ADM-RBAC	The ADM-RBAC approach is an extension of the Ariadne Development Method (ADM), which is a development model for Web systems. ADM divides the development model of Web systems into three phases: conceptual design, detailed design, and evaluation. ADM-RBAC extends ADM with several visual models that specify the role-based access control.
٣٠	AMF (Assurance Management Framework)	The multilayered AMF approach is based on the assurance management framework that focuses on the development of the authorization system. AMF facilitates comprehensive realization of formal security models, security policy specifications, verifications, security code generation, and conformance testing. This multilayered approach includes four development phases: authorization security requirements, authorization model and policy verification, authorization system design, implementation using UML class diagram and OCL constraints, and conformance testing.
٣١	Georg-Aspect Oriented	The Georg-Aspect Oriented approach is based on aspect orientation for designing a secure system. It models authentication protocols and

		possible attacks using UML class and sequence diagrams. By weaving an attack into system models, which results in a so-called misuse model, it can be investigated whether the system is vulnerable to attack. A vulnerable system can be mitigated by weaving an authentication protocol into its design to create a security-treated system model.
٣٢	Gomaa-UML	The Gomaa-UML approach describes a way of modeling complex application designs and requirements in separate ways from modeling security requirements and designs using UML notation, as in use cases, class diagrams for static modeling, and collaboration diagrams. It also separates business concerns from security concerns to reduce the complexity of the requirements and make it possible to maintain the system.
٣٣	Xu- Petri Nets	The threat-driven Xu- Petri Nets approach models the intended functionality of a system and possible threats using Petri nets, whereas mitigations are modeled using Petri net-aspects. Petri nets are a well-studied formal method with graphical and mathematical notations for specifications and analysis of distributed systems. Petri nets can serve as a unified formal basis for specifying system functions, security threats, and threat mitigations. They are expressive in threat modeling.
٣٤	YU-AC	The YU-AC approach uses UML class diagrams augmented with OCL constraints to model role-based access control policies. Scenarios for

		verifying modeled policies are generated from operation invocation patterns. These patterns are manually defined by a designer, and they constrain the initial state and allow the sequence of operation invocations. The patterns are manually created using the best available domain expertise and experience related to the sequences of operations that are likely to uncover policy violations. Each generated scenario needs to be labeled either legal or illegal, and the policy must accept or reject them accordingly. Automatic algorithms for generating scenarios are proposed.
٣٥	KAOS (Keep All Object Satisfied with Intentional Anti-Model)	KAOS is a security requirement driven approach for specifying, analyzing, and modeling application security requirements, and from these security requirements, security design specifications are derived. Also, from the design specifications, the secure code is generated using B method. This approach extends KAOS to include the elaboration of security requirements using anti-models. An anti-model is constructed using obstacles, and an obstacle negates existing goals of the system.
٣٦	HTTP Unit	HTTP unit is a programmable API to detect SQL injection vulnerabilities that help the tester emulate the browser in such a way that the input form could be accessed and modified as a test case. This test case can be checked for the presence of vulnerabilities. This approach is especially for Web applications with agile development.
٣٧	Giordano-	The Giordano-Access Control approach proposes a set of visual

	Access Control	languages to model role-based access policies. The visual languages are intended to be usable by a broad range of users, from developers to top-level managers. Furthermore, XACML policies can be generated from the visual specifications. XACML is an XML-based language for creating access policies and automating their use in the management of access controls for general devices. A group of models for supporting access control can be used in this approach, including the supported models Role, Permission, Separation of Duties, and Role Assignment Diagram. This approach can be embedded into software engineering methodologies for specifying access control policies to be enforced during the design level of information systems or applications.
٣٨	Kim-Access Control	The Kim-Access Control approach combines feature modeling and UML modeling to incorporate role-based access control policies into application models. Feature models define RBAC using UML class and sequence diagrams. These feature models are composited into application models to define domain-specific RBAC policies. The author presents two case studies for the sake of illustration, one a banking system and the other a database management system.
٣٩	Mariscal-AC	The Mariscal-AC approach proposes several extensions of the UML class diagram to model access control policies. The secure sub-system diagram models the public interface that is subject to access controls. The role-slice diagram models the role hierarchy and specifies the

		allowed and disallowed operations for each role. The user diagram models the assignment of users to roles, and the delegation diagram models how users may delegate their roles. Access control policies are modeled separate from the application design to maintain a clear separation of concerns. The approach also provides mapping between the modeled policy and the resulting policy and enforcement codes to allow tracing and enable code generation. Case studies in the university system with prototypes are discussed.
٤٠	Vela-DB-XML	The Vela-DB-XML approach extends UML to model access controls and logging policies for data warehouses. Tagged values in a class diagram indicate security levels, such as top secret or confidential, and roles, such as administrative or passenger. More complex rules, such as log all frustrated access attempts, are modeled as classes. These platform-independent models are transformed to platform-specific models, which can be transformed to implementations for specific databases.
٤١	UML State Charts	The UML state charts approach is a combination of abstract state machine language (ASML) and UML for specific attack scenarios in the requirement specification phase. These attack specifications (scenarios) can be transformed to Snort rules and all of these scenarios are used with some extensions for system intrusion detections.
٤٢	SRS-Tool	This approach is a security requirement process for development

		security requirement specifications with a supported tool called SRS-Tool. It is based on CC and problem profiles and consists of four steps, as follows: Step 1: Analysis of SCL (security classification level) for organization Step 2: Analysis of security environment Step 3: Analysis of security requirement Step 4: Generation of SRS
٤٣	Buyens-LP	The Buyens-LP approach allows the analysis of software architecture for least privilege and separation of duty violations. Based on the architecture and its documentation, a Task Execution Model (TEM) is derived. The TEM identifies the relations between principals and the tasks the system can perform, as represented by the policy defined in the architecture. The analysis consists of verifying whether the TEM is consistent with the intended policy as defined by the requirements.
٤٤	Hoisl-SOA	Hoisl-SOA is a model-driven approach that extends UML activity diagrams, SoaML, and UML4SOA to incorporate security into process-driven, service-oriented architectures (SOA). The UML activity diagram is extended with SecurePin, SecureDataStoreNode, and SecureActivityParameterNode elements to represent secure object flows at the business. Similarly, SoaML and UML4SOA are extended to represent secure object flows at the service level. These models can

		be transformed using another step to Web service artifacts, such as WSDLs. The SoaML provides essential modeling primitives for structural views of a service architecture, including participants, collaborations, service contracts, interfaces, and messages. The UML4SOA extension is used for modeling macroflow/microflow specifications for the participants of a service architecture.
٤٥	UML-AC	The UML-AC approach allows the modeling of access control policies using UML class and object diagrams. A UML class diagram called a type diagram specifies the entities available for modeling the policies and their relations. For example, for a view-based access control (VBAC) policy, these entities are object, permission, role, subject, and view. Object diagrams are used to graphically model policy rules and constraints, and each view must have at least one permission. The provided formal semantics based on graphs allows users to analyze modeled policies by verifying whether all reachable states or policy configurations satisfy all the specified constraints. If this is not the case, the developer must alter the policy rules accordingly.
٤٦	UMLS	The UMLS approach extends several UML elements with labels that specify access control information, such as ownership and read permissions for data. The extended UML diagrams can be transformed to Jif skeleton code. At this level, the Jif compiler can validate the modeled policy.

٤٧	ASF (Agile Security Framework)	The ASF approach helps developers by providing step-by-step guidance for applying security techniques to achieve a secure software system. Also, it introduces security practices at each phase and suggests security training for all developers and stakeholders. It includes hybrid techniques that are a combination of abuser stories and attack trees. Different phases have been proposed in this approach, as follows: 1. Security requirement analysis and planning 2. Threat modeling and designing 3. Secure code implementation 4. Secure deployment
٤٨	ISDF (Integrated Security Development Framework)	The ISDF approach is an integration of carefully selected security patterns into the appropriate stages of the software development lifecycle to ensure the security designs are correctly implemented. A pattern describes a time-tested generic solution to a recurring problem within a specific context. This framework consist of two components. The first is secure development best practice, and the second is a four-stage security pattern. Then the integration between the two components is done for security purposes.
٤٩	EUC (Essential Use Cases)	The processes of the EUC approach begin after the requirement

		engineer gathers the requirements from the stakeholders. The collected requirements are in the form of textual natural language requirements. This approach is for supporting and analyzing the capturing process of security requirements. Also, it supports capturing security requirements of normal business expressed in natural text. Tool support for the sake of applying this approach uses three library patterns: security essential use case, security essential interaction, and security control pattern. The process starts when the textual requirements are analyzed and traced to the EUC patterns library for appropriate abstract interaction in a form of EUC model (1). Then SecEUC are derived from the generated EUC models based on the categorization of their attributes related to the security elements, as defined in the SecEUC pattern library (2). Each SecEUC is mapped to the EUI pattern library (3) for the generation of an abstract prototype in the form of an EUI model. Each EUI model is verified with a defined mandatory security control in the SecCtrl library pattern (4). Next, a recommendation for a graphical user interface (GUI) is provided to visualize the security requirements based on the generated SecEUC (5). This helps ensure the consistency and the correctness of the captured security requirements with the original business requirements provided by the end-user.
٥٠	STS-Tool	The STS-Tool approach is a modelling and analysis support tool for STS-ml, an actor- and goal-oriented security requirement modeling

		language for socio-technical systems. Socio-technical systems consist of social actors, such as humans or organizations, and technical sub-systems in which they interact to achieve their objectives. STS-ml includes high-level organizational concepts, such as actor, goal, and delegation. It is a diagrammatical language that uses graphical concepts and relations to create the models. It also allows modeling with multi-view modeling that includes the social view, information view, and authorization view. The modeling activities consist of five phases: 1. Model of social view 2. Information view 3. Authorization view 4. Automated analysis 5. Deriving the security requirement
٥١	Gupta-Framework	Gupta framework is a security engineering process that converts security requirements and threat into design decisions to mitigate the identified security threats. In this approach, different security requirements are mapped to different security services. The identified design attributes are prioritized, and a security design template is prepared. Based on the final design decision, the appropriate cryptography techniques are chosen from a prepared repository.

٥٢	AEGIS	AEGIS is a secure software development process that concentrates on security requirement specifications through identifying assets and risk analysis. Four design sessions have been proposed between the developers and stakeholders: 1. Software assets and their relationships need to be modeled to identify the security properties to associate with the assets using abuse cases 2. Identify software vulnerabilities, threats, and risk 3. Remove identified vulnerabilities using appropriate security requirements 4. Use other tools, such as static analysis and code review, in the implementation phase
٥٣	AsmL	AsmL is a specification language that is an extension of finite state machine used for representing security requirements. Also, an attack with multiple steps can be captured easily using this approach and presenting them as Snort rules.
٥٤	FDD (Feature Driven Development)	The FDD approach is an agile process for secure Web applications. It integrates agile feature-driven development processes with risk analysis for building secure Web applications. For risk analysis, this approach accesses different paths that could lead to possible attacks and suggests security controls for each possible exploitation of the

		different vulnerabilities. At each increment, the added assets are identified, and the potential attacks are specified. Various security activities are added to the original model to provide security.

List of Publication Venues

Publication Venue	Type	No.	%
ICSE 2007. 29th International Conference on Software Engineering, 2007.	**Conference**	**6**	5.08
ICSE Workshop on Software Engineering for Secure Systems, 2009. SESS '09.	**Conference**	**5**	4.24
Proceedings. Eighth IEEE International Conference on Engineering of Complex Computer Systems, 2002.	Conference	4	3.39
COMPSAC '08. 32nd Annual IEEE International Computer Software and Applications, 2008.	**Conference**	**4**	3.39
Software & Systems Modeling	**Journal**	**4**	3.39
The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.	Conference	3	2.54
15th International Symposium on Software Reliability Engineering, 2004. ISSRE 2004.	Symposium	3	2.54
Information and Software Technology	Journal	3	2.54
IEEE Transactions on Software Engineering.	Journal	2	1.69
Security & Privacy, IEEE	Journal	2	1.69
20th Annual Computer Security Applications Conference, 2004.	Conference	2	1.69
2010 IEEE International Conference on Services Computing (SCC	Conference	2	1.69
ITNG '09. Sixth International Conference on Information Technology: New Generations, 2009.	Conference	2	1.69
Information Systems	Journal	2	1.69
Science of Computer Programming	Journal	2	1.69

Computers & Security	Journal	2	1.69
Model Engineering, Concepts, and Tools 5th International Conference Dresden, Germany, September 30 – October 4, 2002 Proceedings	Lecture Note in computer Science	2	1.69
International Journal of Information Security	Journal	2	1.69
Requirements Engineering	Journal	2	1.69
Communications in Computer and Information Science	Book chapter	2	1.69
Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International	Conference	1	0.85
Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003.	Conference	1	0.85
International Conference on Software Engineering Advances, 2007. ICSEA 2007	Conference	1	0.85
Proceedings. 37th International Conference on Technology of Object-Oriented Languages and Systems,	Conference	1	0.85
ECBS 2006. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, 2006.	Symposium	1	0.85
VL/HCC 2008. IEEE Symposium on Visual Languages and Human-Centric Computing, 2008.	Symposium	1	0.85
IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,	Journal	1	0.85
Proceedings Agile Conference, 2005.	Conference	1	0.85
ICACIA 2008. International Conference on Apperceiving Computing and Intelligence Analysis, 2008.	Conference	1	0.85
11th IEEE High Assurance Systems Engineering Symposium, 2008. HASE 2008.	Conference	1	0.85
QSIC '08. The Eighth International Conference on Quality Software, 2008.	Conference	1	0.85
Proceedings. 2006 31st IEEE Conference on Local Computer Networks	Conference	1	0.85
ICACT 2008. 10th International Conference on Advanced Communication Technology, 2008.	Conference	1	0.85
Proceedings First IEEE International Workshop on Source Code Analysis and Manipulation	Workshop	1	0.85
IEEE Symposium on Security and Privacy, 2006	Symposium	1	0.85
Software, IEEE	Journal	1	0.85
2006 Eighth IEEE International Symposium on Web Site	Symposium	1	

Evolution (WSE'06)			0.85
Proceedings. International Conference on Dependable Systems and Networks, 2002.	Conference	1	0.85
12th Working Conference on Reverse Engineering (WCRE'05)	Conference	1	0.85
7th IEEE International Conference on Computer and Information Technology (CIT 2007)	Conference	1	0.85
Third International Symposium on Information Assurance and Security 2007. IAS 2007	Symposium	1	0.85
2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)	Conference	1	0.85
2012 IEEE Sixth International Conference on Software Security and Reliability (SERE)	Conference	1	0.85
Computer Standards & Interfaces	Journal	1	0.85
Journal of Visual Languages & Computing	Journal	1	0.85
Journal of Systems and Software	Journal	1	0.85
Decision Support Systems	Journal	1	0.85
Computers & Operations Research	Journal	1	0.85
Journal of Advanced Research	Journal	1	0.85
International Journal on Software Tools for Technology Transfer	Journal	1	0.85
11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006	Book chapter	1	0.85
4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings	Book chapter	1	0.85
Computational Science and Its Applications – ICCSA 2005	Lecture Note in computer Science	1	0.85
Lecture Notes in Business Information Processing	Book chapter	1	0.85
Computer Security – ESORICS 2003	Book chapter	1	0.85
Proceedings 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008	Lecture Note in computer Science	1	0.85
Proceedings International Conference, ICAC3 2011, Mumbai, India, January 28-29, 2011.	Book chapter	1	0.85
Advances in Information Security and Its Application	Book chapter	1	0.85

Proceedings First Asia Pacific Requirements Engineering Symposium, APRES 2014, Auckland, New Zealand, April 28-29, 2014.	Book chapter	1	0.85
Lecture Notes in Computer Science	Book chapter	1	0.85
CSI Transactions on ICT	Journal	1	0.85
Proceedings 23rd IFIP WG 6.1 International Conference, ICTSS 2011, Paris, France, November 7-10, 2011.	Lecture Note in Computer Science	1	0.85
Proceedings of the eighth ACM symposium on Access control models and technologies - SACMAT '03	Symposium	1	0.85
NSPW '03 Proceedings of the 2003 workshop on New security paradigms	Workshop	1	0.85
International Journal of Electronic Security and Digital Forensics	Journal	1	0.85
PST '06 Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services	Conference	1	0.85
Proceedings of the 6th international conference on Web engineering - ICWE '06	Conference	1	0.85
Proceedings of the 2008 international symposium on Software testing and analysis ISSTA '08	Symposium	1	0.85
Proceedings of the 15th international conference on World Wide Web WWW '06	Conference	1	0.85
Proceedings of the 44th annual southeast regional conference on - ACM-SE 44	Conference	1	0.85
Proceedings of the 2007 workshop on Programming languages and analysis for security - PLAS '07	Conference	1	0.85
USENIX-SS'06 Proceedings of the 15th conference on USENIX Security Symposium	Conference	1	0.85
Proceedings of the 14th ACM conference on Computer and communications security - CCS '07	Conference	1	0.85
Proceedings of the third international workshop on Dynamic analysis - WODA '05	Workshop	1	0.85
SAC '07 Proceedings of the 2007 ACM symposium on Applied computing	Symposium	1	0.85
SOSP '07 Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles	Symposium	1	0.85
LCSD '07 Proceedings of the 2007 Symposium on Library-Centric Software Design	Symposium	1	0.85
POPL '07 Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages	Conference	1	0.85
EuroSys '09 Proceedings of the 4th ACM European	Conference	1	

conference on Computer systems			0.85
Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12	Conference	1	0.85
iiWAS '11 Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services	Conference	1	0.85
Quality and Reliability Engineering International	Journal	1	0.85
Total		**118**	100.00

References

[1] M. U. A. Khan and M. Zulkernine, "On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software," *2009 33rd Annu. IEEE Int. Comput. Softw. Appl. Conf.*, pp. 353–358, 2009.

[2] J. Jurjens, *Secure Systems Development with UML*. Springer, Berlin., 2004.

[3] G. McGraw, *Software Security: Building Security In,*. Addison Wesley, 2006.

[4] G. McGraw, "Testing for security during development: why we should scrap penetrate-and-patch," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 13, no. 4, pp. 13–15, 1998.

[5] Benjamin Fabian, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requir. Eng.*, vol. 15, no. 1, pp. 7–40, 2010.

[6] M. Hadavi, V. Hamishagi, and H. Sangchi, "Security Requirements Engineering; State of the Art and Research Challenges," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2008, vol. 8, pp. 19–21.

[7] P. Karpati, Y. Redda, A. L. Opdahl, and G. Sindre, "Comparing attack trees and misuse cases in an industrial setting," *Inf. Softw. Technol.*, vol. 56, no. 3, pp. 294–308, 2014.

[8] M. U. a. Khan and M. Zulkernine, "On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software," in *2009 33rd Annual IEEE International Computer Software and Applications Conference*, 2009, vol. 2, pp. 353 – 358.

[9] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Comput. Stand. Interfaces*, vol. 32, no. 4, pp. 153–165, 2010.

[10] I. A. Tondel, M. G. Jaatun, and P. H. Meland, "Security Requirements for the Rest of Us: A Survey," *Software, IEEE*, vol. 5, no. 1, pp. 20 – 27, 2008.

[11] B. Musa Shuaibu, N. Md Norwawi, M. H. Selamat, and A. Al-Alwani, "Systematic review of web application security development model," *Artif. Intell. Rev.*, vol. 23, no. 2, pp. 259–276, 2013.

[12] I. Ghani and I. Yasin, "Software Security Engineering In Extreme Programming Methodology: A Systematic Literature," *Sci. Int.*, vol. 25, no. 2, pp. 215–221, 2013.

[13] F. Roeser, “Can Software Security be Successfully Implemented in Agile Software Development? A Systematic Literature Review,” *Florian Roeser Hochschule der Medien. Nobelstr. 10, 70569 Stuttgart, Ger.*, pp. 1–20, 2010.

[14] B. A. and S. C. Kitchenham, “Guidelines for performing Systematic Literature Reviews in Software Engineering,” 2007.

[15] G. McGraw, “Software Security,” *Secur. Privacy, IEEE*, vol. 2, no. 2, pp. 80–83, 2004.

[16] G. Mcgraw, “From the ground up: the DIMACS software security workshop,” *Secur. Privacy, IEEE*, vol. 1, no. 2, pp. 59 – 66, 2007.

[17] G. McGraw and G. Hoglund, *Exploiting Software: How to Break Code*. Boston: Addison- Wesley, 2004.

[18] D. Verdon and G. McGraw, “Risk analysis in software design,” *IEEE Secur. Priv. Mag.*, vol. 2, no. 4, pp. 79–84, 2004.

[19] C. P. S. Pfleeger., *Security in Computing*. Ptr: Prentice Hall, 2006.

[20] Jayaram K RAditya P Mathur, “Software Engineering for Secure Software - State of the Art : A Survey Technical report, Purdue University,” 2005.

[21] C. E. . Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, “A taxonomy of computer program security flaws,” *ACM Comput. Surv.*, vol. 26, no. 3, pp. 211–254, 1994.

[22] M. U. a Khan and M. Zulkernine, “Quantifying security in secure software development phases,” in *32nd Annual IEEE International Computer Software and Applications, 2008. COMPSAC ’08.*, 2008, pp. 955–960.

[23] G. McGraw, “Building secure software: better than protecting bad software’,” *Software, IEEE*, vol. 19, no. December, pp. 57–59, 2002.

[24] M. A. Hadavi, H. Shirazi, H. M. Sangchi, and V. S. Hamishagi, “Software Security; A Vulnerability Activity Revisit,” *2008 Third Int. Conf. Availability, Reliab. Secur.*, pp. 866–872, Mar. 2008.

[25] P. N. P. B. W. Boehm, “Understanding and controlling software costs - Software Engineering, IEEE Transactions on,” *IEEE Trans. Softw. Eng.*, vol. 14, no. 10, pp. 1462–1477, 1990.

[26] M. Umair, A. Khan, and M. Zulkernine, “A Survey on Requirements and Design Methods for Secure Software Development *,” Ontario, Canada, 2009.

[27] J. V. and G. McGraw, *Building Secure Software*. Addison- Wesley, 2002.

[28] J. Romero-Mariona, H. Ziv, and D. J. Richardson, "Formality of the Security Specification Process: Benefits Beyond Requirements," *2010 43rd Hawaii Int. Conf. Syst. Sci.*, pp. 1–6, 2010.

[29] J. Gr, K. Buyens, B. De Win, R. Scandariato, W. Joosen, C. Science, K. U. Leuven, and B.- Leuven, "On the Secure Software Development Process : CLASP and SDL Compared," in *Proceeding SESS '07 Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, 2007, p. 1.

[30] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requir. Eng.*, vol. 10, no. 1, pp. 34–44, Jun. 2004.

[31] J. Romero-mariona and D. J. Richardson, "SRRS : A Recommendation System for Security Requirements," pp. 50–52, 2008.

[32] G. McGraw, "A Portal for Software Security," *Secur. Priv. Mag. IEEE*, vol. 3, no. 4, pp. 75–79, 2005.

[33] G. Mcgraw, "Managing software security risks," *Computer (Long. Beach. Calif).*, vol. 35, no. 4, pp. 99–101, 2002.

[34] M. Howard and S. Lipner, "Inside the windows security push," *IEEE Secur. Priv.*, vol. 1, pp. 57–61, 2003.

[35] M. Howard, "Building more secure software with improved development processes," *IEEE Secur. Priv. Mag.*, vol. 2, no. 6, pp. 63–65, 2004.

[36] W. PETERS, J. F. & PEDRYCZ, *Software engineering : an engineering approach*. Wiley., 2000.

[37] J. A. O'Brien, *Management Information Systems: Managing Information Technology in the EBusiness Enterprise*. New York: McGraw Hill., 2002.

[38] J. Britton, C. & Doake, *Software System Development. A general introduction*. London :McGraw Hill., 2003.

[39] D. L. Post, J. & Anderson, *Management information systems: Solving business problems with information technology*. McGraw Hill, 2003.

[40] L. A. Maciaszek, *Requirements Analysis and System Design*. Addison Wesley, 2001.

[41] L. I. Futrell, R. T., Shafer, D. F. & Shafer, *Quality software project management*. Prentice Hall, 2002.

[42] H. Van Vliet, *Software engineering - principles and practice*. New York: John Wiley & Sons, 2002.

[43] 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.

[44] R. L. Jones and A. Rastogi, "Secure Coding: Building Security into the Software Development Life Cycle," *Inf. Syst. Secur.*, vol. 13, no. 5, pp. 29–39, Nov. 2004.

[45] R. Breu, K. Burger, M. Hafner, and G. Popp, "Towards a Systematic Development of Secure Systems," *Inf. Syst. Secur.*, vol. 13, no. 3, pp. 5–13, May 2004.

[46] L. Futcher and R. Von Solms, "SecSDM : A Model for Integrating Security into the Software Development Life Cycle," in *Proceedings of the 5 th World Conference on Information Security Education*, vol. 237, 2007, pp. 41–48.

[47] L. A. Futcher, "an integrated risk-based approach to support it undergraduate students in secure software development," Nelson Mandela Metropolitan University, 2011.

[48] M. I. Daud, "Secure software development model: A guide for secure software life cycle," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2010, vol. I.

[49] B. Taylor and S. Azadegan, "Threading secure coding principles and risk analysis into the undergraduate computer science and information systems curriculum," in *Proceedings of the 3rd annual conference on Information security curriculum development - InfoSecCD '06*, 2006, p. 24.

[50] S. Lipner, "The Trustworthy Computing Security Development Lifecycle," in *20th Annual Computer Security Applications Conference*, 2004, pp. 2–13.

[51] G. Sindre and A. L. Opdahl, "Eliciting security requirements by misuse cases," in *2000. TOOLS-Pacific 2000. Proceedings. 37th International Conference on Technology of Object-Oriented Languages and Systems*, 2000, vol. 10, no. 1, pp. 34–44.

[52] R. Hassan, S. Bohner, S. El-Kassas, and M. Eltoweissy, "Goal-Oriented, B-Based Formal Derivation of Security Design Specifications from Security Requirements," in *Third International Conference on Availability, Reliability and Security, 2008. ARES 08.*, 2008, pp. 1443–1450.

[53] B. Best, J. Jürjens, B. Nuseibeh, W. Hall, and M. Keynes, "Model-based Security Engineering of Distributed Information Systems using UMLsec," in *ICSE 2007.*

29th International Conference on Software Engineering, 2007., 2007, pp. 581 – 590.

[54] a. Apvrille and M. Pourzandi, "Secure Software Development by Example," *IEEE Secur. Priv. Mag.*, vol. 3, no. 4, pp. 10–17, Jul. 2005.

[55] J. Jürjens and P. Shabalin, "Tools for secure systems development with UML," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, no. 5–6, pp. 527–544, Jul. 2007.

[56] T. Lodderstedt and D. Basin, "SecureUML : A UML-Based Modeling Language for Model-Driven Security," in *Proceedings Model Engineering, Concepts, and Tools 5th International Conference Dresden, Germany, September 30 – October 4, 2002*, 2002, pp. 426–441.

[57] Jan Jürjens, "UMLsec : Extending UML for Secure Systems Development," in *Model Engineering, Concepts, and Tools 5th International Conference Dresden, Germany, September 30 – October 4, 2002 Proceedings*, 2002, pp. 412–425.

[58] I. Alexander, "Initial industrial experience of misuse cases in trade-off analysis," *Proc. IEEE Jt. Int. Conf. Requir. Eng.*, 2002.

[59] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, 1999, pp. 55–64.

[60] P. Salini and S. Kanmani, "Survey and analysis on Security Requirements Engineering," *Comput. Electr. Eng.*, vol. 38, no. 6, pp. 1785–1797, 2012.

[61] D. Mu, V. Chiprianov, Laurent Gallon, and Philippe Aniort, "A Review of Security Requirements Engineering Methods with Respect to Risk Analysis and Model-Driven Engineering," in *Lecture Notes in Computer Science*, 2014, pp. 79–93.

[62] J. Du, Y. Ye, and Q. Wang, "An analysis for understanding software security requirement methodologies," in *SSIRI 2009 - 3rd IEEE International Conference on Secure Software Integration Reliability Improvement*, 2009, pp. 141–149.

[63] R. Matulevičius and M. Dumas, "A Comparison of SecureUML and UMLsec for Role-based Access Control," in *Proceedings of the 9th Conference on Databases and Information Systems*, 2010, pp. 171–185.

[64] C. Raspotnig and A. Opdahl, "Comparing risk identification techniques for safety and security requirements," *J. Syst. Softw.*, vol. 86, no. 4, pp. 1124–1151, 2013.

[65] S. Dasanayake, J. Markkula, and M. Oivo, “Concerns in Software Development – A Systematic Mapping Study,” in *Proceedings - 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 14–17.

[66] J. Webster and R. T. Watson, “Analyzing the past to prepare for the future: Writing a literature review,” *Manag. Inf. Syst. Q. 26.2 3*, vol. 26, no. 2, 2003.

[67] C. Wohlin, “Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering,” in *EASE ’14 Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014.

[68] L. Lin, D. Ince, W. Hall, M. K. Mk, E. L. C. Lin, B. Nuseibeh, D. C. Ince, and M. Jackson, “Using Abuse Frames to Bound the Scope of Security Problems,” in *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, 2004, pp. 3–4.

[69] C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh, “Security Requirements Engineering: A Framework for Representation and Analysis,” *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 133–153, Jan. 2008.

[70] C. B. Haley, W. Hall, J. D. Moffett, and R. Laney, “A Framework for Security Requirements Engineering,” in *SESS ’06 Proceedings of the 2006 international workshop on Software engineering for secure systems*, 2006, pp. 35–41.

[71] K. R. van Wyk and G. McGraw, “Bridging the Gap between Software Development and Information Security,” *Secur. Privacy, IEEE*, vol. 3, no. 5, pp. 75–79, Sep. 2005.

[72] M. Kainerstorfer, J. Sametinger, and A. Wiesauer, “Software Security for Small Development Teams – A Case Study,” in *iiWAS ’11 Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, 2011, pp. 5–7.

[73] D. P. Gilliam, T. L. Wolfe, J. S. Sherif, and M. Bishop, “Software security checklist for the software life cycle,” *WET ICE 2003. Proceedings. Twelfth IEEE Int. Work. Enabling Technol. Infrastruct. Collab. Enterp. 2003.*, pp. 243–248, 2003.

[74] M. Essafi, L. Labed, and H. Ben Ghezala, “S2D-ProM: A Strategy Oriented Process Model for Secure Software Development,” in *International Conference on Software Engineering Advances (ICSEA 2007)*, 2007, no. Icsea, pp. 24–24.

[75] G. Sindre and A. L. Opdahl, “Eliciting security requirements with misuse cases,” *Requir. Eng.*, vol. 10, no. 1, pp. 34–44, Jun. 2004.

[76] M. Hussein and M. Zulkernine, "UMLintr: a UML profile for specifying intrusions," in *13th Annual IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)*, 2006, p. 8 pp.–288.

[77] M. Raihan, M. Zulkernine, "AsmLSec : An Extension of Abstract State Machine Language for Attack," in *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.*, 2007, pp. 775 – 782.

[78] P. Diaz, I. Aedo, D. Sanz, and A. Malizia, "A model-driven approach for the visual specification of Role-Based Access Control policies in web systems," *IEEE Symp. Vis. Lang. Human-Centric Comput. 2008. VL/HCC 2008.*, pp. 203–210, 2008.

[79] H. Hu and G. Ahn, "Constructing Authorization Systems Using Assurance Management Framework," *IEEE Trans. Syst. Man, Cybern. Part C Appl. Rev.*, vol. 40, no. 4, pp. 396–405, 2010.

[80] G. Georg, F. Collins, I. Ray, and R. France, "Using Aspects to Design a Secure System," in *Proceedings. Eighth IEEE International Conference on Engineering of Complex Computer Systems,2002.*, 2002, pp. 117 – 126.

[81] H. Gomaa, "Modeling Complex Systems by Separating Application and Security Concerns," in *Proceedings. Ninth IEEE International Conference on Engineering Complex Computer Systems, 2004.*, 2004, pp. 19 – 28.

[82] M. Menzel and C. Meinel, "SecureSOA Modelling Security Requirements for Service-Oriented Architectures," *2010 IEEE Int. Conf. Serv. Comput.*, pp. 146–153, Jul. 2010.

[83] M. Menzel and C. Meinel, "A Security Meta-model for Service-Oriented Architectures," *2009 IEEE Int. Conf. Serv. Comput.*, pp. 251–259, 2009.

[84] D. Informatics and T. U. Munich, "Sound Methods and Effective Tools for Model-based Security Engineering with UML," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005, pp. 322 – 331.

[85] J. Jürjens, M. Lehrhuber, G. Wimmel, and T. U. München, "Model-Based Design and Analysis of Permission-Based Security," in *Proceedings. 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. ICECCS 2005.*, 2005.

[86] J. Jan, P. Bartmann, and S. Jörg, "Model-based Security Analysis for Mobile Communications," in *ICSE '08 Proceedings of the 30th international conference on Software engineerin*, 2008, vol. 2, pp. 683–692.

[87] D. Xu, S. Member, and K. E. Nygard, “Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets,” *IEEE Trans. Softw. Eng.*, vol. 32, no. 4, pp. 265–278, 2006.

[88] L. Yu, R. France, I. Ray, and S. Ghosh, “A Rigorous Approach to Uncovering Security Policy Violations in UML Designs,” *2009 14th IEEE Int. Conf. Eng. Complex Comput. Syst.*, pp. 126–135, 2009.

[89] a. Tappenden, P. Beatty, J. Miller, a. Geras, and M. Smith, “Agile security testing of Web-based systems via HTTPUnit,” *Agil. Dev. Conf.*, pp. 29–38, 2005.

[90] H. Shahriar and M. Zulkernine, “Mutation-Based Testing of Buffer Overflow Vulnerabilities,” *COMPSAC ’08. 32nd Annu. IEEE Int. Comput. Softw. Appl. 2008.*, pp. 979–984, 2008.

[91] X.-S. Zhang, Lin Shao, and Jiong Zheng, “A NOVEL METHOD OF SOFTWARE VULNERABILITY DETECTION BASED ON FUZZING TECHNIQUE,” in *ICACIA 2008. International Conference on Apperceiving Computing and Intelligence Analysis, 2008.*, 2008, pp. 270–273.

[92] H. Shahriar and M. Zulkernine, “Mutation-Based Testing of Format String Bugs,” *11th IEEE High Assur. Syst. Eng. Symp. 2008. HASE 2008*, pp. 229–238, Dec. 2008.

[93] M. Junjin, “An Approach for SQL Injection Vulnerability Detection,” *Sixth Int. Conf. Inf. Technol. New Gener. 2009. ITNG ’09.*, pp. 1411–1414, 2009.

[94] A. Kie, P. J. Guo, and M. D. Ernst, “Automatic Creation of SQL Injection and Cross-Site Scripting Attacks,” in *IEEE 31st International Conference on Software Engineering, 2009. ICSE 2009.*, 2009, pp. 199–209.

[95] H. Shahriar and M. Zulkernine, “MUSIC: Mutation-based SQL Injection Vulnerability Checking,” *QSIC ’08. Eighth Int. Conf. Qual. Software, 2008.*, pp. 77–86, Aug. 2008.

[96] W. Allen, C. Dou, and G. Marin, “A Model-based Approach to the Security Testing of Network Protocol Implementations,” *Proceedings. 2006 31st IEEE Conf. Local Comput. Networks*, pp. 1008–1015, Nov. 2006.

[97] H. Kim, Y. Choi, D. Lee, and D. Lee, “Practical Security Testing using File Fuzzing,” *ICACT 2008. 10th Int. Conf. Adv. Commun. Technol. 2008.*, vol. 2, pp. 1304–1307, Feb. 2008.

[98] J. Offutt, “Bypass Testing of Web Applications,” *15th Int. Symp. Softw. Reliab. Eng. 2004. ISSRE 2004.*, pp. 187–197, 2004.

[99] H. Shahriar and M. Zulkernine, “MUTEC : Mu tation-based Te sting of C ross Site Scripting School of Computing,” in *ICSE Workshop on Software Engineering for Secure Systems, 2009. SESS ’09.*, 2009, pp. 47–53.

[100] C. Del Grosso, G. Antoniol, E. Merlo, and P. Galinier, “Detecting buffer overflow via automatic test input data generation,” *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3125–3143, Oct. 2008.

[101] S. Mcallister, E. Kirda, and C. Kruegel, “Leveraging User Interactions for In-Depth Testing of Web Applications,” in *Proceedings 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008.*, 2008, pp. 191–210.

[102] S. F. Hidhaya and A. Geetha, “Intrusion Protection against SQL Injection and Cross Site Scripting Attacks Using a Reverse Proxy,” in *Communications in Computer and Information Science*, 2012, pp. 252–263.

[103] A. Mammar, A. Cavalli, W. Jimenez, W. Mallouli, and E. M. De Oca, “Using Testing Techniques for Vulnerability Detection in C Programs,” in *Proceedings 23rd IFIP WG 6.1 International Conference, ICTSS 2011, Paris, France, November 7-10, 2011.*, 2011, pp. 80–96.

[104] R. Xu, P. Godefroid, and R. Majumdar, “Testing for Buffer Overflows with Length Abstraction,” in *Proceedings of the 2008 international symposium on Software testing and analysis ISSTA ’08*, 2008, pp. 27–37.

[105] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, “SecuBat : A Web Vulnerability Scanner,” in *Proceedings of the 15th international conference on World Wide Web WWW ’06*, 2006, pp. 247–256.

[106] W. Du and A. P. Mathur, “Testing for software vulnerability using environment perturbation,” *Qual. Reliab. Eng. Int.*, vol. 18, no. 3, pp. 261–272, May 2002.

[107] M. Weber, V. Shah, and C. Ren, “A case study in detecting software security vulnerabilities using constraint optimization,” *Proc. First IEEE Int. Work. Source Code Anal. Manip.*, pp. 1–11, 2001.

[108] N. Jovanovic, C. Kruegel, and E. Kirda, “Pixy : A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper),” in *IEEE Symposium on Security and Privacy, 2006*, 2006, p. 6 pp. – 263.

[109] G. Wassermann and Z. Su, “Static detection of cross-site scripting vulnerabilities,” in *30th International Conference on Software Engineering, 2008. ICSE ’08.*, 2008, p. 171.

[110] D. Evans, D. ;Larochelle, "Improving Security Using Extensible Lightweight Static Analysis," *Software, IEEE*, vol. 19, no. 1, pp. 42 – 51, 2002.

[111] J. Zhu, J. Xie, H. R. Lipford, and B. Chu, "Supporting secure programming in web applications through interactive static analysis," *J. Adv. Res.*, vol. 5, no. 4, pp. 449–462, Jul. 2014.

[112] B. Hackett, M. Das, D. Wang, and Z. Yang, "Modular Checking for Buffer Overflows in the Large," in *ICSE '06 Proceedings of the 28th international conference on Software engineering*, 2006, pp. 232–241.

[113] J.-E. J. Tevis and J. a. Hamilton, "Static analysis of anomalies and security vulnerabilities in executable files," in *Proceedings of the 44th annual southeast regional conference on - ACM-SE 44*, 2006, p. 560.

[114] K. Chen and D. Wagner, "Large-scale analysis of format string vulnerabilities in Debian Linux," in *Proceedings of the 2007 workshop on Programming languages and analysis for security - PLAS '07*, 2007, p. 75.

[115] Y. Xie, "Static Detection of Security Vulnerabilities in Scripting Languages," in *USENIX-SS'06 Proceedings of the 15th conference on USENIX Security Symposium*, 2006, vol. 15, pp. 147–160.

[116] G. Agosta, A. Barenghi, A. Parata, and G. Pelosi, "Automated Security Analysis of Dynamic Web Applications through Symbolic Code Execution," *2012 Ninth Int. Conf. Inf. Technol. New Gener. (ITNG)*, pp. 189–194, Apr. 2012.

[117] H. AL-Amro and E. El-Qawasmeh, "Discovering security vulnerabilities and leaks in ASP.NET websites," in *2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, 2012, pp. 329–333.

[118] A. Mohosina and M. Zulkernine, "DESERVE: A Framework for Detecting Program Security Vulnerability Exploitations," *2012 IEEE Sixth Int. Conf. Softw. Secur. Reliab.*, pp. 98–107, Jun. 2012.

[119] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, "Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis," *(COMPSAC), 2012 IEEE 36th Annu. Comput. Softw. Appl. Conf.*, pp. 233–243, Jul. 2012.

[120] A. Aggarwal and P. Jalote, "Integrating Static and Dynamic Analysis for Detecting Vulnerabilities," *30th Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 343–350, 2006.

[121] M. Muthuprasanna, K. Wei, and S. Kothari, "Eliminating SQL Injection Attacks - A Transparent Defense Mechanism," in *2006 Eighth IEEE International Symposium on Web Site Evolution (WSE'06)*, 2006, pp. 22–32.

[122] M. Monga, R. Paleari, and E. Passerini, "A Hybrid Analysis Framework for Detecting Web Application Vulnerabilities," in *SESS '09. ICSE Workshop on Software Engineering for Secure Systems, 2009.*, 2009, pp. 25–32.

[123] D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna, "Multi-module vulnerability analysis of web-based applications," in *Proceedings of the 14th ACM conference on Computer and communications security - CCS '07*, 2007, p. 25.

[124] W. G. J. Halfond and A. Orso, "Combining static analysis and runtime monitoring to counter SQL-injection attacks," in *Proceedings of the third international workshop on Dynamic analysis - WODA '05*, 2005, pp. 1–7.

[125] M. Johns and C. Beyerlein, "SMask : Preventing Injection Attacks in Web Applications by Approximating Automatic Data / Code Separation," in *SAC '07 Proceedings of the 2007 ACM symposium on Applied computing*, 2007, pp. 284–291.

[126] T. Tsai and N. Singh, "Libsafe: transparent system-wide protection against buffer overflow attacks," *Proceedings. Int. Conf. Dependable Syst. Networks, 2002. DSN 2002.*, p. 541, 2002.

[127] I. C. Computacionales, C. Scientifique, and M. Bât, "AProSec : an Aspect for Programming Secure Web Applications," in *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.*, 2007, no. 1.

[128] S. Chong, J. Liu, A. C. Myers, X. Qi, K. V. Lantian, and Z. Xin, "Secure Web Applications via Automatic Partitioning," in *SOSP '07 Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, 2007, pp. 31–44.

[129] N. Juillerat, "Enforcing Code Security in Database Web Applications Using Libraries and Object Models," in *LCSD '07 Proceedings of the 2007 Symposium on Library-Centric Software Design*, 2007, pp. 31–41.

[130] J. Xie, H. Lipford, and B.-T. Chu, "Evaluating interactive support for secure programming," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*, 2012, p. 2707.

[131] H. Nishiyama, "SecureC: control-flow protection against general buffer overflow attack," *29th Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 149–155, 2005.

[132] J. R. Cordy and T. R. Dean, "Enhancing Security Using Legality Assertions," *12th Work. Conf. Reverse Eng.*, no. 2, pp. 35–44, 2005.

[133] E. Ofuonye and J. Miller, "Resolving JavaScript Vulnerabilities in the Browser Runtime," *2008 19th Int. Symp. Softw. Reliab. Eng.*, pp. 57–66, Nov. 2008.

[134] D. Yu, A. Chander, N. Islam, and I. Serikov, "JavaScript instrumentation for browser security," in *POPL '07 Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2007, vol. 42, no. 1, p. 237.

[135] J.-C. Lin and J.-M. Chen, "The Automatic Defense Mechanism for Malicious Injection Attack," *7th IEEE Int. Conf. Comput. Inf. Technol. (CIT 2007)*, pp. 709–714, Oct. 2007.

[136] A. Smirnov and T. Chiueh, "Automatic Patch Generation for Buffer Overflow Attacks," *Third Int. Symp. Inf. Assur. Secur. 2007. IAS 2007*, pp. 165–170, Aug. 2007.

[137] F. Dysart and M. Sherriff, "Automated Fix Generator for SQL Injection Attacks," *2008 19th Int. Symp. Softw. Reliab. Eng.*, pp. 311–312, Nov. 2008.

[138] Q. Gao, W. Zhang, Y. Tang, and F. Qin, "First-Aid : Surviving and Preventing Memory Management Bugs during Production Runs," in *EuroSys '09 Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 159–172.

[139] D. Mellado, E. Fernández-Medina, and M. Piattini, "A common criteria based security requirements engineering process for the development of secure information systems," *Comput. Stand. Interfaces*, vol. 29, no. 2, pp. 244–253, Feb. 2007.

[140] D. Mellado and E. Fern, "Applying a Security Requirements Engineering Process," in *Proceedings 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006.*, 2006, pp. 192–206.

[141] D. Mellado, E. Fernández-medina, and M. Piattini, "Security Requirements Management in Software Product Line Engineering," in *Communications in Computer and Information Science*, 2009, pp. 250–263.

[142] H. Mouratidis, P. Giorgini, and G. Manson, "When security meets software engineering: a case of modelling secure information systems," *Inf. Syst.*, vol. 30, no. 8, pp. 609–629, Dec. 2005.

[143] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Requirements engineering for trust management: model, methodology, and reasoning," *Int. J. Inf. Secur.*, vol. 5, no. 4, pp. 257–274, Aug. 2006.

[144] L. Dai and K. Cooper, “Using FDAF to bridge the gap between enterprise and software architectures for security,” *Sci. Comput. Program.*, vol. 66, no. 1, pp. 87–102, Apr. 2007.

[145] L. Dai and K. Cooper, “Modeling and performance analysis for security aspects,” *Sci. Comput. Program.*, vol. 61, no. 1, pp. 58–71, Jun. 2006.

[146] M. Giordano, G. Polese, G. Scanniello, and G. Tortora, “A system for visual role-based policy modelling,” *J. Vis. Lang. Comput.*, vol. 21, no. 1, pp. 41–64, Feb. 2010.

[147] S. Kim, D.-K. Kim, L. Lu, S. Kim, and S. Park, “A feature-based approach for modeling role-based access control systems,” *J. Syst. Softw.*, vol. 84, no. 12, pp. 2035–2052, Dec. 2011.

[148] J. a. Pavlich-Mariscal, S. a. Demurjian, and L. D. Michel, “A framework of composable access control features: Preserving separation of access control concerns from models to code,” *Comput. Secur.*, vol. 29, no. 3, pp. 350–379, May 2010.

[149] J. Trujillo, E. Soler, E. Fernández-Medina, and M. Piattini, “An engineering process for developing Secure Data Warehouses,” *Inf. Softw. Technol.*, vol. 51, no. 6, pp. 1033–1051, Jun. 2009.

[150] E. Fernández-Medina, J. Trujillo, R. Villarroel, and M. Piattini, “Developing secure data warehouses with a UML extension,” *Inf. Syst.*, vol. 32, no. 6, pp. 826–856, Sep. 2007.

[151] E. Fernández-Medina and M. Piattini, “Designing secure databases,” *Inf. Softw. Technol.*, vol. 47, no. 7, pp. 463–477, May 2005.

[152] J. Abramov, A. Sturm, and P. Shoval, “Evaluation of the Pattern-based method for Secure Development (PbSD): A controlled experiment,” *Inf. Softw. Technol.*, vol. 54, no. 9, pp. 1029–1043, Sep. 2012.

[153] J. Abramov, O. Anson, M. Dahan, P. Shoval, and A. Sturm, “A methodology for integrating access control policies within database development,” *Comput. Secur.*, vol. 31, no. 3, pp. 299–314, May 2012.

[154] B. Vela, C. Blanco, E. Fernández-Medina, and E. Marcos, “A practical application of our MDD approach for modeling secure XML data warehouses,” *Decis. Support Syst.*, vol. 52, no. 4, pp. 899–925, Mar. 2012.

[155] M. Clavel, V. Silva, C. Braga, and M. Egea, “Model-Driven Security in Practice : An Industrial Experience,” in *Lecture Notes in Computer Science*, 2008, pp. 326–337.

[156] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security for process-oriented systems," *Proc. eighth ACM Symp. Access Control Model. Technol. - SACMAT '03*, p. 100, 2003.

[157] M. Zulkernine, M. Graves, and M. U. A. Khan, "Integrating software specifications into intrusion detection," *Int. J. Inf. Secur.*, vol. 6, no. 5, pp. 345–357, May 2007.

[158] S. Choi, S. Chae, and G. Lee, "SRS-Tool : A Security Functional Requirement Specification Development Tool for Application Information System of Organization SRS-Process : A Development Process for SRS," in *Computational Science and Its Applications – ICCSA 2005*, 2005, pp. 458–467.

[159] M. Memon, G. D. Menghwar, M. H. Depar, A. a. Jalbani, and W. M. Mashwani, "Security modeling for service-oriented systems using security pattern refinement approach," *Softw. Syst. Model.*, vol. 13, no. 2, pp. 549–572, Aug. 2012.

[160] R. Breu, M. Hafner, F. Innerhofer-Oberperfler, and F. Wozak, "Model-Driven Security Engineering of Service Oriented Systems," in *Lecture Notes in Business Information Processing*, 2008, p. pp 59–71.

[161] K. Buyens, R. Scandariato, and W. Joosen, "Least privilege analysis in software architectures," *Softw. Syst. Model.*, vol. 12, no. 2, pp. 331–348, Nov. 2011.

[162] B. Hoisl, S. Sobernig, and M. Strembeck, "Modeling and enforcing secure object flows in process-driven SOAs : an integrated model-driven approach," *Softw. Syst. Model.*, pp. 513–548, 2014.

[163] M. Koch and F. Parisi-Presicce, "UML specification of access control policies and their formal verification," *Softw. Syst. Model.*, vol. 5, no. 4, pp. 429–447, Oct. 2006.

[164] R. Heldal and F. Hultin, "Bridging Model-Based and Language-Based Security," in *Computer Security – ESORICS 2003*, 2003, pp. 235–252.

[165] A. Singhal, "Development of Agile Security Framework Using a Hybrid Technique for Requirements Elicitation," in *Proceedings International Conference, ICAC3 2011, Mumbai, India, January 28-29, 2011.*, 2011, pp. 178–188.

[166] A. Alkussayer and W. H. Allen, "The ISDF Framework : Integrating Security Patterns and Best Practices," in *Advances in Information Security and Its Application*, Springer, 2009, pp. 17–28.

[167] S. Yahya, M. Kamalrudin, S. Sidek, and J. Grundy, "Capturing Security Requirements Using Essential Use Cases (EUCs)," in *Proceedings First Asia*

Pacific Requirements Engineering Symposium, APRES 2014, Auckland, New Zealand, April 28-29, 2014., 2014, pp. 16–30.

[168] E. Paja, F. Dalpiaz, and P. Giorgini, "STS-Tool : Security Requirements Engineering for Socio-Technical Systems," in *Lecture Notes in Computer Science*, 2014, pp. 65–96.

[169] K. Chatterjee, D. Gupta, and A. De, "'A framework for development of secure software,'" *CSI Trans. ICT*, vol. 1, no. 2, pp. 143–157, Mar. 2013.

[170] H. Suleiman and D. Svetinovic, "Evaluating the effectiveness of the security quality requirements engineering (SQUARE) method : a case study using smart grid advanced metering infrastructure," *Requir. Eng.*, pp. 251–279, 2013.

[171] N. R. Mead and T. Stehney, "Security quality requirements engineering (SQUARE) methodology," *SESS '05 Proc. 2005 Work. Softw. Eng. Secur. Syst. Trust. Appl.*, pp. 1–7, Jul. 2005.

[172] J. Viega, "Building Security Requirements with CLASP," in *SESS '05 Proceedings of the 2005 workshop on Software engineering for secure systems—building trustworthy applications*, 2005, pp. 1–7.

[173] I. Flechais, C. Mascolo, and M. A. Sasse, "INTEGRATING SECURITY AND USABILITY INTO THE REQUIREMENTS AND DESIGN PROCESS," *Int. J. Electron. Secur. Digit. Forensics*, vol. 1, no. 1, pp. 12–26, 2007.

[174] I. Fle, M. A. Sasse, and S. M. V Hailes, "Bringing Security Home : A process for developing secure and usable systems," in *NSPW '03 Proceedings of the 2003 workshop on New security paradigms*, 2003, pp. 49–57.

[175] M. Graves, "Bridging the Gap : Software Specification Meets Intrusion Detector," in *PST '06 Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, 2006, pp. 1–8.

[176] X. Ge, R. F. Paige, F. a. C. Polack, H. Chivers, and P. J. Brooke, "Agile development of secure web applications," in *Proceedings of the 6th international conference on Web engineering - ICWE '06*, 2006, p. 305.

[177] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository," *Inf. Softw. Technol.*, vol. 55, no. 10, pp. 1679–1694, 2013.

[178] T. Okubo and H. Tanaka, "Identifying Security Aspects in Early Development Stages," *2008 Third Int. Conf. Availability, Reliab. Secur.*, pp. 1150–1157, 2008.

[179] M. Talib, Adel Khelifi, and Leon Jololian, "Secure Software Engineering: A New Teaching Perspective Based on the SWEBOK," *Interdiscip. J. Information, Knowledge, Manag.*, vol. 5, pp. 83–99, 2010.

[180] A. Souag, "Towards a New Generation of Security Requirements Definition Methodology Using Ontologies," *24th Int. Conf. Adv. Inf. Syst. Eng.*, 2012.

[181] Y. Alotaibi and F. Liu, "How to Model a Secure Information System (IS): A Case Study," *Int. J. Inf. Educ. Technol.*, vol. 2, no. 2, pp. 94–102, 2012.

[182] a Youseef and F. Liu, "A New Framework to Model a Secure E-Commerce System," *World Acad. Sci. Eng. Technol.*, vol. 6, no. 1, pp. 6–12, 2012.

[183] C. Y. Lester and F. Jamerson, "Incorporating software security into an undergraduate software engineering course," in *Proceedings - 2009 3rd International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2009*, 2009, pp. 161–166.

[184] G. Elahi, "Security Requirements Engineering: State of the Art and Practice and Challenges," 2009.

[185] H. H. Albreiki and Q. H. Mahmoud, "Evaluation of Static Analysis Tools for Software Security," in *10th International Conference on Innovations in Information Technology (INNOVATIONS), 2014*, 2014, pp. 93–98.

[186] K. Schneider, E. Knauss, S. Houmb, S. Islam, and J. Jürjens, "Enhancing security requirements engineering by organizational learning," *Requir. Eng.*, vol. 17, no. 1, pp. 35–56, 2012.

[187] O. Daramola, Y. Pan, P. Karpati, and G. Sindre, "A comparative review of i*-based and use case-based security modelling initiatives," *Proc. - Int. Conf. Res. Challenges Inf. Sci.*, 2012.

[188] N. Ikram, S. Siddiqui, and N. F. Khan, "Security Requirement Elicitation Techniques : The Comparison of Misuse Cases and Issue Based Information Systems," in *2014 IEEE Fourth International Workshop on Empirical Requirements Engineering (EmpiRE)*, 2014, pp. 36–43.

[189] J. Heikka, "Abuse Cases Revised : An Action Research Experience 2 . Existing Research on Abuse and Misuse Cases," in *The Tenth Pacific Asia Conference on Information Systems (PACIS 2006)*, 2006, no. Pacis, pp. 673–684.

[190] M. N. Johnstone, "Modelling misuse cases as a means of capturing security requirements," *Proc. 9th Aust. Inf. Secur. Manag. Conf.*, pp. 140–147, 2011.

[191] F. Massacci and F. Paci, “How to Select a Security Requirements Method? A Comparative Study with Students and Practitioners,” *Proc. 17th Nord. Conf. Secur. {IT} Syst.*, pp. 89–104, 2012.

[192] K. Labunets, F. Massacci, F. Paci, and L. M. S. Tran, “An experimental comparison of two risk-based security methods,” *Int. Symp. Empir. Softw. Eng. Meas.*, pp. 163–172, 2013.

[193] K. Labunets, “Empirical Validation of Security Methods,” in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013.

[194] P. L. P. Li and B. C. B. Cui, “A comparative study on software vulnerability static analysis techniques and tools,” *Inf. Theory Inf. Secur. (ICITIS), 2010 IEEE Int. Conf.*, 2010.

[195] K. Beznosov and B. Chess, “Security for the Rest of Us: An Industry Perspective on the Secure-Software Challenge,” *Software, IEEE*, vol. 25, no. 1, pp. 10–12, 2008.

[196] T. Okubo, K. Taguchi, H. Kaiya, and N. Yoshioka, “MASG: Advanced Misuse Case Analysis Model with Assets and Security Goals,” *J. Inf. Process.*, vol. 22, no. 3, pp. 536–546, 2014.

[197] S. Hedayatpour, N. Kama, and S. Chuprat, “Analyzing Security Aspects during Software Design Phase using Attack-based Analysis Model,” *Int. J. Softw. Eng. Its Appl.*, vol. 8, no. 3, pp. 143–156, 2014.

[198] C. C. R. Busby-earle, R. B. France, and I. Ray, “Analysing Requirements to Detect Latent Security Vulnerabilities,” in *(SERE-C), 2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*, 2014, pp. 168 – 175.

[199] M. Riaz, J. Slankas, J. King, L. Williams, and N. Carolina, “Using Templates to Elicit Implied Security Requirements from Functional Requirements A Controlled Experiment,” in *ESEM ’14 Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 18–19.

[200] M. Riaz, J. Slankas, J. King, L. Williams, and N. Carolina, “Using Templates to Elicit Implied Security Requirements from Functional Requirements A Controlled Experiment,” in · *Proceeding ESEM ’14 Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 18–19.

[201] Y. Alotaibi, “Business process modelling challenges and solutions: a literature review,” *J. Intell. Manuf.*, 2014.

[202] Y. Alotaibi and F. Liu, “A novel secure business process modeling approach and its impact on business performance,” *Inf. Sci. (Ny).*, vol. 277, pp. 375–395, 2014.

[203] P. H. Nguyen, J. Klein, Y. Le Traon, M. E. Kramer, and Y. Le Traon, “A Systematic Review of Model-Driven Security,” *2013 20th Asia-Pacific Softw. Eng. Conf.*, pp. 432–441, 2013.

[204] G. Díaz and J. R. Bermejo, “Static analysis of source code security: Assessment of tools against SAMATE tests,” *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1462–1476, 2013.

[205] Z. Zhioua, S. Short, and Y. Roudier, “Static Code Analysis for Software Security Verification: Problems and Approaches,” *2014 IEEE 38th Int. Comput. Softw. Appl. Conf. Work.*, pp. 102–109, 2014.

[206] M. K. Gupta, M. C. Govil, and G. Singh, “An Approach to Minimize False Positive in SQLI Vulnerabilities Detection Techniques through Data Mining,” in *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT)*, 2014, pp. 407–410.

[207] M. Kulenovic and D. Donko, “A survey of static code analysis methods for security vulnerabilities detection,” *2014 37th Int. Conv. Inf. Commun. Technol. Electron. Microelectron.*, no. May, pp. 1381–1386, 2014.

[208] V. Rafael, L. De Mendonça, and C. L. Rodrigues, “Static Analysis Techniques and Tools : A Systematic Mapping Study,” *CSEA 2013 Eighth Int. Conf. Softw. Eng. Adv.*, no. c, pp. 72–78, 2013.

[209] H. Shahriar and M. Zulkernine, “Mitigating program security vulnerabilities: Approaches and challenges,” *ACM Comput. Surv.*, vol. 44, no. 3, pp. 1–46, 2012.

[210] M. Lapke, “InjectIng SecurIty Into InformatIon SyStemS Development,” *Port. J. Manag. Stud.*, pp. 235–248, 2010.

[211] H. Mouratidis, “Secure Software Systems Engineering: The Secure Tropos Approach,” *J. Softw.*, vol. 6, no. 3, 2011.

[212] T. Li and J. Horkoff, “Dealing with security requirements for socio-technical systems: A holistic approach,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8484 LNCS, pp. 285–300, 2014.

[213] H. Mouratidis, A. Sunyaev, and J. Jurjens, “Secure Information Systems Engineering: Experiences and Lessons Learned from Two Health Care Projects,” *21st Int. Conf. CAiSE 2009, Amsterdam, Netherlands, June 8-12, 2009. Proc.*, pp. 231–245, 2009.

[214] N. Argyropoulos, "Designing secure software systems Combining goal-oriented modeling and risk management," Utrecht University, 2014.

[215] N. Mead, "How to compare the Security Quality Requirements Engineering (SQUARE) method with other methods," *C. SEI*, no. August, 2007.

[216] J. Romero-Mariona, H. Ziv, and D. J. Richardson, "Formality of the security specification process: Benefits beyond requirements," *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, pp. 1–6, 2010.

[217] J. Romero-Mariona, H. Ziv, and D. Richardson, "SRRS: a recommendation system for security requirements," *Proc. 2008 Int. Work. Recomm. Syst. Softw. Eng.*, pp. 50–52, 2008.

[218] L. Rostad, "An extended misuse case notation: Including vulnerabilities and the insider threat," *Twelfth Work. Conf. Requir. Eng. Found. Softw. Qual.*, pp. 67–77, 2006.

[219] M. Diallo and J. Romero-Mariona, "A comparative evaluation of three approaches to specifying security requirements," *Proc. Int. Work. Conf. Requir. Eng. Found. Software, Qual. (REFSQ'06)*, 2006.

[220] S. Heckman and L. Williams, "A systematic literature review of actionable alert identification techniques for automated static code analysis," *Inf. Softw. Technol.*, vol. 53, no. 4, pp. 363–387, 2011.

[221] D. A. Ableidinger, "SECURITY IN SOFTWARE DEVELOPMENT, Why Security is Lacking in Software," The College of St. Scholastica, 2013.

[222] L. Levi, Qin Zhangb, and Yves Le Traonb, "Advances in Model-Driven Security," in *Contribution to collective works*, 2013, p. 59.

Vitae

Name :Nabil Mohammed Abdo Mohammed

Nationality :Yemeni

Date of Birth :5/1/1983

Email :g200905310@kfupm.edu.sa

Address Taiz - Yemen

Academic Background **:**

- Received Bachelor of Computer Education– King Khalid University (KKU) – Abha, Saudi Arabia , 2008.
- Joint the Information and Computer science department as full time student at King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in September 2010.
- Completed Master of science (M.S.) in Computer science from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in May 2015.