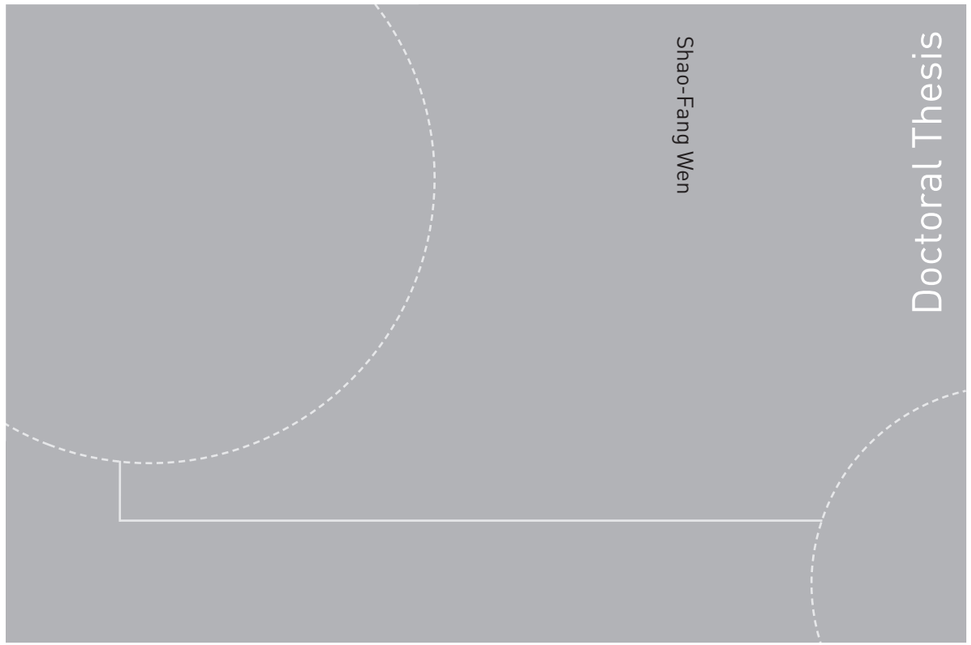
ISBN 978-82-326-4650-0 (printed version)
ISBN 978-82-326-4651-7 (electronic version)
ISSN 1503-8181



Doctoral theses at NTNU, 2020:151

Shao-Fang Wen

A Multi-Discipline Approach for Enhancing Developer Learning in Software Security

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology
and Electrical Engineering
Department of Information Security
and Communication Technology

NTNU
Norwegian University of
Science and Technology

Doctoral theses at NTNU, 2020:151

NTNU

NTNU
Norwegian University of
Science and Technology

Shao-Fang Wen

A Multi-Discipline Approach for Enhancing Developer Learning in Software Security

Thesis for the degree of Philosophiae Doctor

Gjøvik, May 2020

Norwegian University of Science and Technology
Faculty of Information Technology
and Electrical Engineering
Department of Information Security and Communication
Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology
and Electrical Engineering
Department of Information Security and Communication
Technology

ISBN 978-82-326-4650-0 (printed version)
ISBN 978-82-326-4651-7 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2020:151

Printed by Skipnes Kommunikasjon as

Dedicated to my beloved parents and family.

Declaration of Authorship

I, Shao-Fang Wen, hereby declare that this thesis and the work presented in it are entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed:

(Shao-Fang Wen)

Date:

Abstract

Building secure software is challenging. Developers should possess proper security knowledge and skills so that they can resist security attacks and implement security countermeasures effectively. However, the lack of knowledge about security among software developers has become a major problem in software communities. Software developers come in the field from different academic disciplines, and many of them lack formal, college-level software development and security training. Even in the curricula of computer science or engineering, educational programs seem to fail at providing students (future developers) with essential knowledge and skills in secure software development. Without appropriate knowledge to resist security attacks and implement corresponding security countermeasures, developers lose the capability to handle the growing complexity of software development, and the software products become more vulnerable to security risks consequently.

To help software developers become aware of the increasing cybersecurity threats, security experts and software practitioners are devoted to offering a large body of security knowledge regarding standards, guidelines, and techniques, which are available in the open literature or on the internet. However, such exponential growth of knowledge resources does not make a considerable contribution to improve the problem of software insecurity. The conventional approaches on security knowledge instruction seem to lose effectiveness in fostering developers' learning of software security. What is more, the contextual factors within software development organizations, technical and non-technical, are influencing developers' learning processes toward the achievement of secure software development. The lack of supportive learning environments in software development, along with ineffective teaching approaches for software security, has created difficulties for developers in learning security knowledge.

This thesis is centered in the discipline of Information System and draws from cross-disciplinary thinking at the intersections of sociology, education, software engineering and others, to undertake the complex task of identifying how to help enhance developers learning in software security. With the goals of investigating contextual factors that affect developers' learning of software security and suggesting a learning tool for effective security education and learning, this thesis contributes to the fields of software development and security education. This thesis employs a five-cycles of Design Science Research (DSR) methodology to apply existing models and means from the theories of socio-technical system and context-based teaching and learning to suggest a multi-discipline approach that integrates necessary elements for the goal achievement. The contribution of the thesis is twofold: First, this thesis offers a conceptual framework to identifying the complex relationship between technical and social factors, pointing out the limitations and opportunities of security learning in software development. The conceptual framework allows software organizations to think holistically about their strategies so that they can undertake the challenges of

secure software development through establishing a supportive security learning environment within the organization. Second, this thesis forges a concrete artifact designed to promote context-based learning of security knowledge: the ontology-based contextualized learning system. Through evaluation in both pedagogical and software development environments, it is proved to contribute a solution to the problem domain. While these results are positive, the innovative context-based artifact benefits not only the domain of software security, but also other educational fields, such as information security and computer security.

Acknowledgment

Research depends on a set of enabling conditions including funding, protected time and encouragement. Without these conditions being so generously available, this work would not have been possible. I am therefore thankful for people's efforts to make this doctoral work a reality. This gratitude is primarily directed towards my supervisors, Stewart Kowalski, Basel Katt, and Rune Hjelsvold, who patiently have guided me through the process with a lot of engagement and encouragement. I do not think I could have done this without your advice, enthusiasm, and support. I am deeply indebted to you.

Besides my supervisors, I would like to express my full appreciation to the staff of the Department of Information Security and Communication Technology (IIK) of NTNU for their administrative support, Nils Karlstad Svendsen, Kathrine Huke Markengbakken, Hilde Bakke, Jingjing Yang, Linda Derawi, Urszula Nowostawska, Ingrid Schantz Bakka, and Marina Shalaginova. Numerous others have also contributed to inspire and challenge my ideas underpinning this work. In particular, I want to thank my NTNU colleagues and friends, Bian Yang, Mariusz Nowostawski, Gaute Wangen, Vasileios Gkioulos, Vivek Agrawal, Adam Szekeres, Mazaher Kianpour, Greth Østby, Muhammad Mudassar Yaminm, and others. I have a fantastic experience spending time in your company.

Most importantly, I must thank my wife, Hung-Pei Chen, who was with me through this entire journey from start to finish. I thank you for the inspiration, encouragement, love, support, and most of all your patience! I also wish to thank my three precious daughters, Shin-Ru, Shin-Rung and Wan-Chi, whose smiles and hugs are a source of endless delight. The most stressful moments are endured better with your laughter. Immense thanks go also to my parents who raised me and taught me to study hard and to give priority in my life to the quest for knowledge. Thanks for your love and blessing.

Last but certainly not least, through this journey, I have met many talented and compassionate individuals who did not hesitate to devote their valuable time to me when it was needed. I dare not risk missing to mention anyone's names, so I will simply say "Thank you ALL for being there for me".

Content

List of Figures

List of Tables

List of Acronyms and Abbreviations

CAPEC	Common Attack Pattern Enumeration and Classification
CBK	Common Body of Knowledge
CBL	Context-Based Learning
CERT	Computer Emergency Response Team
CIA	Confidentiality, Integrity, and Availability
CoP	Community of Practice
CSIS	Center for Strategic and International Studies
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposures
CVS	Concurrent Versions System
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DC	Design Cycle
DHS	United States Department of Homeland Security
DoS	Denial-of-Service
DSR	Design Science Research
GPL	General Public License
KM	Knowledge Management
ICT	Information and Communication Technology
IDE	Integrated Development Environment
IM4TD	Idealized Model for Theory Development
IMR	Introduction, Methodology and Result
IS	Information System
IT	Information Technology
MDA	Model Driven Architecture

MMR	Mixed-Method Research
NIST	National Institute of Standards and Technology
NTNU	Norwegian University of Science and Technology
NVD	National Vulnerability Database
OBTL	Outcome-Based Teaching and Learning
OSD	Open Source Definition
OSS	Open Source Software
OSSC	Open Source Software Community
OWASP	Open Web Application Security Project
OWL	Web Ontology Language
PoC	Proof-of-Concept
QaR	Qualitative Research
QnR	Quantitative Research
RDF	Resource Description Framework
RP	Research Paper
RQ	Research Question
SAMM	Software Assurance Maturity Model
SDLC	Software Development Lifecycle
SLR	Systematic Literature Review
SPARQL	Protocol and RDF Query Language
SPC	Secure Programming Clinic
SQL	Sequential Query Language
SQLi	SQL Injection
SSDLC	Secure Software Development Lifecycle
SSL	Secure Socket Layer
STACK	Security Toolbox: Attacks & Countermeasures
STS	Socio-Technical System
XSS	Cross-Site Scripting

Part I

Introductory Chapters

Chapter 1

Introduction

This chapter offers contexts of the research before presenting the problem description, the motivation for the research, and the research questions. Furthermore, it provides an overview of related research publications and their relationship to this thesis's research questions. Lastly, a thesis outline is presented.

1.1 Research Context

In the modern world, information and communication technology (ICT) is broadly used as a tool or facilitator supporting the development of society in general. Society heavily relies on ICT to carry out daily activities such as manipulating and storing personal information, health records, financial transactions, and other sensitive information. Software, as a dominant factor in the development of ICT systems, plays a crucial role in the entire ICT value chain, including the platform, network, and device. According to a forecast by Gartner, Inc., worldwide ICT spending was projected to total $3.8 trillion in 2019, with software products and services representing nearly 33% of that figure. The software has developed over time to fit changing needs; for example, people can connect with each other easily through the internet. However, as software becomes increasingly complex and connected, it also features many more flaws for hackers to exploit [341]. A global report by the Center for Strategic and International Studies[1] and McAfee[2] [273] has stated that close to $600 billion is lost to cybercrime each year. Some of the most widespread software-based

[1] The Center for Strategic and International Studies, based in Washington, D.C. (United States), conducts policy studies and performs strategic analyses on political, economic, and security issues throughout the world.

[2] McAfee is a U.S.-based global security technology company and part of the Intel Security division.

crimes include stealing information via hacking, carrying out virus attacks to cripple computer systems, and implanting spyware with the intent of watching people perform computer activities. In the age of cybercrime, and with threats to software on the rise and attacks increasingly complex, the importance of not only application security (e.g., encryption, firewalls, and access control) but also software security[3] has been recognized [295].

In this era of information explosion, numerous possibilities exist to become a software developer, regardless of one's background and expertise. According to the 2018 Stack Overflow[4] developer survey [426], of the more than 10,000 participating developers, one-third were from other academic disciplines, such as natural science, mathematics, and business disciplines, while nearly 90% of respondents reported that they were self-taught about programming skills. With internet technologies, people enjoy easy access to many sorts of information helpful for learning and practicing software programming; they can even release their software products for public use or distribute software codes among broad communities of developers around the world. Yet, only a small fraction of developers are competent at secure software development[5]. Many computer science courses such as programming and system development leave software security out of their mandatory curricula [380, 511], while software security is an optional discipline. A survey by Veracode [6] and DevOps.com[7] [261] found that only 2.8% of undergraduate computer science programs require a security course, while only 24% of 397 respondents, who were college-educated developers, were required to complete cybersecurity courses as part of their education. What is more, 70% of the respondents said that the security education they received was not adequate for what their job positions required [261].

To emphasize software security, security researchers and software practitioners have mounted substantial efforts towards providing guidelines, standards, or frameworks for secure software development, which are available in open literature or on the internet [155, 194, 453, 510]. Such works have resulted in the creation of a huge body of security knowledge[8] that developers can learn and refer to. Nevertheless, the

[3] Software security is "the idea of engineering software so that it continuous to function correctly under malicious attacks" [293]. The concept of software security is introduced in Section 2.1.1.
[4] Stack Overflow is an online community for people interested in learning to code and sharing their knowledge regarding software development: https://stackoverflow.com
[5] Secure software development encompasses the security-related methods to an existing software development process. The details about secure software development are given in Section 2.1.3.
[6] Veracode is a service provider of enterprise-class application security, integrating agile security solutions for organizations around the globe.
[7] DevOps.com collects original content related to DevOps on the web, including philosophy, tools, business impact, and best practices.
[8] The terms "security knowledge," "secure software knowledge," and "software security knowledge" are used as inclusive terms in this thesis. They all refer to knowledge of engineering software that allows one to ensure that software continues to function correctly under malicious attacks. The details are discussed in the Section 2.1.4.

number of new vulnerabilities in software systems has continued to increase. According to Common Vulnerabilities and Exposures (CVE)[9] vulnerability statistics [102]—available in Figure 1.1—2.5 times more software vulnerabilities were disclosed in 2018 than in 2010. The 2018 figure represents an all-time high of 16,555 vulnerabilities, with almost 45 vulnerabilities reported on an average day. Of these vulnerabilities, nearly 70% were due to programming errors; the rest were due to configuration or design problems [256]. Despite the fact that vulnerabilities have been a focus of the security community for years, a substantial majority of the vulnerabilities were classic and fairly well-known programming errors. Such errors, including cross-site scripting (XSS) and injection flaws, have been repeatedly reported and have appeared on the OWASP[10] Top 10 vulnerabilities list every year since 2010 [495], and nearly 80% of recently scanned applications still suffer from such issues [457].

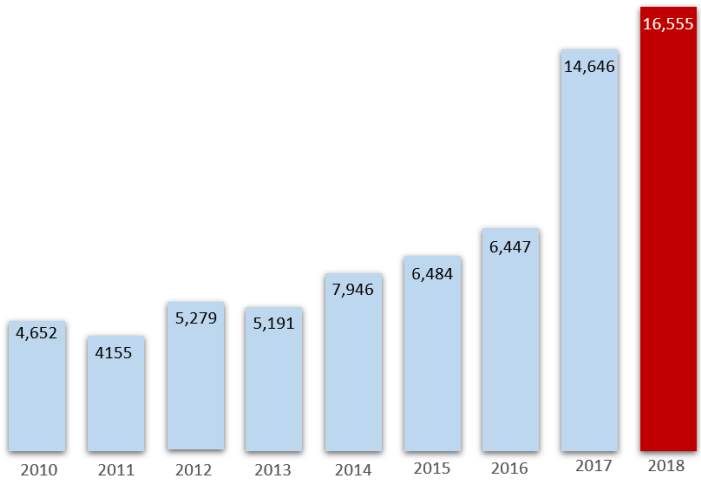


Figure 1.1: The number of security-related vulnerabilities registered in the Common Vulnerabilities and Exposures system from 2010 to 2018 [102]

1.2 Research Problem

One of the major problems in software security is the lack of knowledge about security among software developers [31, 430, 438]. Building secure software is challenging: technologies advance rapidly, and the growing intricacy of ICT systems has made all software projects quite different in terms of context and development techniques [294]. Such complexity means that developers should possess proper

[9]The CVE system provides a reference-method for publicly known information security vulnerabilities and exposures: https://cve.mitre.org/.

[10] The Open Web Application Security Project (OWASP) is an online community that produces freely available articles, methodologies, documentation, and technologies in the field of web application security: https://www.owasp.org/

security knowledge and skills so that they can resist security attacks and implement security countermeasures effectively [46]. However, software developers are not experts in security in general [2, 387]. Many of them come in the field from other academic disciplines and have no formal, college-level software development and security training. Even in the curricula of computer science or software engineering, education programs seem to fail at providing students with essential competence in software security.

To help developers stay on the cutting edge, security communities and industries are devoted to offering a substantial amount of security learning materials in the form of checklists, standards, and best practices; developers can access these materials via books, open literature, or the internet. However, without fundamental security education, developers lack capabilities to sort out the complex and scattered pieces of security information and to distinguish between relevance and irrelevance. Such exponential growth in learning materials has also created excessive amount of information, leading to a heavy cognitive load for learners [508], which makes it difficult for them to learn the required subjects quickly and conveniently from various sources [208]. Consequently, the attitude of learners towards learning generally declines during the progression through learning sessions because of the overloaded state [317]. Further, in the conventional learning materials[11], instructions commonly start with abstract security concepts, as opposed to being situated in real-life contexts. Learners who learn security concepts solely in a decontextualized setting might not be able to apply the necessary skills when facing real-life security threats [384] or with the feeling that secure software development is so difficult to achieve that they simply cast it aside [23].

Another important consideration related to the difficulty of acquiring security knowledge is the learning environment that surrounds developers, which relates to the culture, business goals, and structures of the organizations. The world of technologies advances constantly, and business requirements are continuously changing. Software development organizations face high pressure regarding productivity and constant demands for innovation and rapid responses to markets. As a result, software developers would typically focus on their programming skills, implementing as many functionalities as possible before their deadline, and they later patch any bugs before the next release or hotfix [175, 218]. Stress and resource fatigue is common among software project teams. Given these social and organizational influence, developers often lack opportunities to reflect on the quality of their code or lack a strong desire to continue learning [262]. In this setting, obtaining security knowledge becomes an occasional activity, which is highly dependent on the learning environment given to developers.

The lack of supportive learning environments in software development, along with ineffective teaching and learning approaches for software security, has created

[11] The weakness of conventional security learning materials is discussed in Section 2.2.2

difficulties in learning essential security knowledge, ranging from basic vulnerabilities to in-depth security practices on secure software development. Consequently, developers fail to possess adequate security knowledge and skills to build secure software. This is attributed to the knowledge gap between where learners are now (initial state) and where they need to be (goal state or solution) [193]. Developers' level of security knowledge acquired from the learning materials is the initial state, and what they need to know to secure their software systems is the goal state. Figure 1.2 hypothetically depicts software system complexity and secure software knowledge as functions over time. The knowledge gap described above is also visible as a function over time. The explosion in security learning materials does not make a considerable contribution to improving the problem. Without appropriate measures to help developers gain security knowledge effectively, the gap will continue to widen.

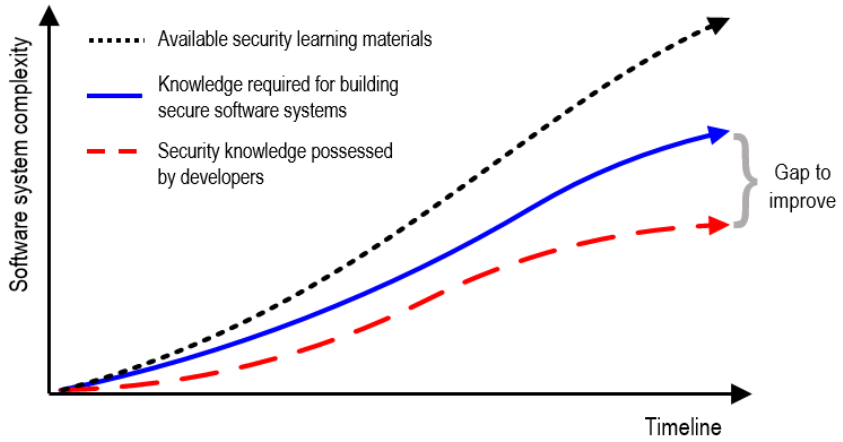


Figure 1.2: The knowledge gap for secure software development

1.3 Research Motivation

Improving software security requires many different approaches. One is to give software developers the knowledge to develop and maintain software programs that handle errors and resist attacks appropriately [46]. Such knowledge makes developers more sensitive to the intimation of security mistakes. However, today's teaching practices and learning materials for software security seem to lose the effectiveness of fostering security learning, either for students or developers. Meanwhile, the social and technical conditions within software development environments are complicating the learning process for developers in terms of security aspects. In the research of computing disciplines[12], the lack of integrative research and the limited use of relevant reference disciplines have been identified

[12] ACM outlines five major disciplines within the computing field: computer engineering, computer science, information systems, information technology and software engineering.

problems for some time [165, 167]. These facts demonstrate a need for a multi-discipline approach, technical and non-technical, for alternative and complementary teaching and learning techniques facilitating a learning environment that offers continuous security education for developers.

To facilitate effective learning[13], researchers have provided a variety of frameworks offering a comprehensive view of general teaching and learning contexts. For example, Biggs [43] developed the Presage-Process-Product (3P) model of learning, emphasizing on the curriculum and course design, which is synthesized with Outcome-Based Teaching and Learning (OBTL). Race's model [362], which suggests there are five factors underpinning successful learning (i.e. wanting, needing, doing, feedback and digesting), drew on ideas emanating from psychology. As this research is concerned with supportive conditions for security learning in software development and the creation of effective learning opportunities, this thesis utilizes Fenstermacher and Richardson's framework [142] in the conceptualization of the research phenomenon. Fenstermacher and Richardson [142] have presented four ingredients that focus on teaching, learning, and their interaction in learning environments: (a) willingness and effort on the part of the learner, (b) social surroundings that are supportive of teaching and learning, (c) opportunities to teach and learn, and (d) good teaching [142]. The four ingredients highlight the value of a setting as a framework within which learners encounter social and content related focal events, determining tasks as opportunities to learn and talk about relevant knowledge, initiating willingness and effort if successfully designed. Fenstermacher and Richardson's framework centers on practices of classroom teaching, however, it has been used to deal with many areas with the science of learning. These areas include online learning practices [121, 285], instructional quality [221, 285], didactics [270], pedagogical content knowledge [12, 96], and learners' interest and motivation [305]. Consequently, this framework offers researchers a platform to study effective security-learning environments, which should be essential to consider the state of the learners (e.g., interest, motivation, and other aspects related to willingness and effort), the character of the social surroundings (e.g., policies, culture, and norms of the groups that support and assist in learning), and the availability and extent of opportunities for learning.

On the one hand, learning could be conceptualized within different contexts and applied to numerous organizational activities related to people, processes, and learning techniques. In the context of software development, software developers collaborate in teams and groups embedded within their work organizations. The activities that developers perform are not only technical tasks but also a social process embedded within organizational and cultural structures [109]. Such socio-technical structures include a wide range of contextual factors with potential influence in terms of guiding developers or inspiring learning, for instance, the security value of the

[13] Effective learning encompasses appropriate approaches and strategies that provide effectiveness for the particular goals and context [477].

organization, peers' expectation and encouragement toward security, and the project structure for secure software development; these factors also lead to the success or failure of software development projects. The socio-technical view of learning focuses on the organizational strategy of harmonizing learning activities with technological drivers and social enablers to achieve objectives [201]. The theory of socio-technical systems embraces the combined social and technical complexity of work organization [128, 442], and it has the explicit ambition of improving peoples' job satisfaction and productivity while simultaneously creating the conditions necessary for an adaptive and learning-centric organization [442]. Researchers have continuously addressed the importance of the social and human side of learning in software engineering [63, 111, 148, 166]; however, the socio-technical perspectives of developer learning in software security have not yet been well examined.

On the other hand, effective teaching techniques require motivation on the part of the learner and opportunities for learning through the provision of appropriate facilities and resources [142]. These features of learning suggest a proposal for developing an engaged learning environment to cultivate learners' intrinsic motivation, which could significantly increase the likelihood of teaching being successful. According to Jonassen and Land [224], "learners must be introduced to the context of the problem and its relevance, and this must be done in a way that motivates and engages them" (p. 33). Context and the particulars of that context can provide a powerful motivation for learning [88]. This thesis recommends that to create opportunities and conditions supporting more effective learning about software security in software development, and to motivate developers to learn about software security, *educators should contextualize security teaching and learning, placing the knowledge in a context familiar to learners*. Context-based approaches[14] aim to bring science learning closer to the lives and interests of learners and to illustrate how using familiar contexts can increase their interest in science and therefore enhance their understanding [38]. Researchers have identified several interrelated problems and challenges in science education and learning that context-based learning approaches intend to address: (a) curricula are overloaded [162, 411], (b) too many isolated facts and concepts prevent students from developing a worthwhile "mental model" [307], and (c) an excessive emphasis on correct explanations and solid foundations leaves students confused about reasons for learning science [162, 323]. As these problems have plagued security education, context-based learning may be relevant as regards software security. This approach is not new, and education researchers have emphasized learning in context over the years; however, such approaches are not embraced in practice in the domain of software security, and much remains to be learned about designing learning support artifacts for use in context-based education.

To overcome the aforementioned problems and socio-technical challenges regarding security learning and the limitations of related research, this thesis addresses the elements of both socio-technical and context-based approaches that are necessary for

[14] Context and Context-Based Learning are discussed in Section 2.3.

security learning to be effective. Figure 1.3 contains a schematic overview of the research problems and motivation.

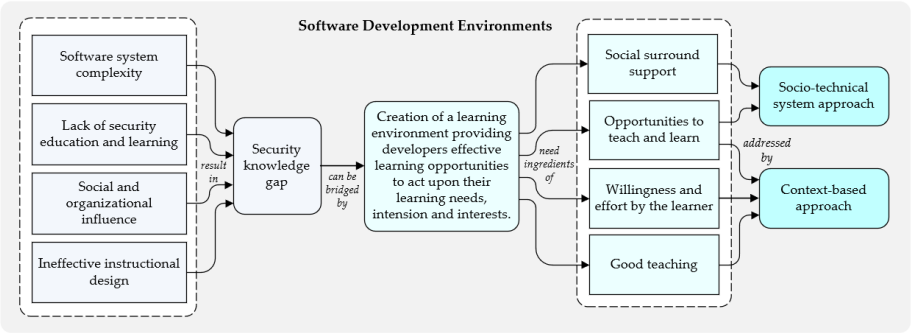


Figure 1.3: A schematic overview of research problems and motivation

1.4 Research Objectives and Research Questions

This thesis is centered in the discipline of Information System (IS)[15] and draws from cross-disciplinary thinking at the intersections of sociology, education, software engineering and others, to undertake the complex task of identifying how to help developers bridge the security knowledge gap. The underlying research relied on multifaceted approaches aimed at expanding the current understanding of security education and learning. Consequently, this thesis aims to accomplish the following objectives: (a) establishing a socio-technical foundation for understanding security learning in the context of software development and (b) proposing an online learning system, restructuring security knowledge and facilitating a context-based learning process to help developers and other learners learn software security. To achieve the research objectives, four main research questions (RQs) were formulated to guide the research activities. Figure 1.4 illustrates the research activities with the corresponding research questions.

RQ 1: How do socio-technical aspects affect individuals' learning of software security in the context of open source software development?

The first research question encourages empirical investigations of the magnitude of the real-world problems in secure software development. This question attempts to identify opportunities, prospectus, and limitations related to learning software security, specifically in open source software (OSS) development environments. In the domain of software development, it is difficult to draw precise and conclusive boundaries regarding what constitutes useful background and what does not. Additionally, for reasons of practicality, investigating all possible sources of influence

[15] Information System disciplines examine topics related largely to organizational concepts, especially technology adoption and operation, all primarily at a behavioral level of analysis. The academic disciplines of this thesis are described in Section 1.5.

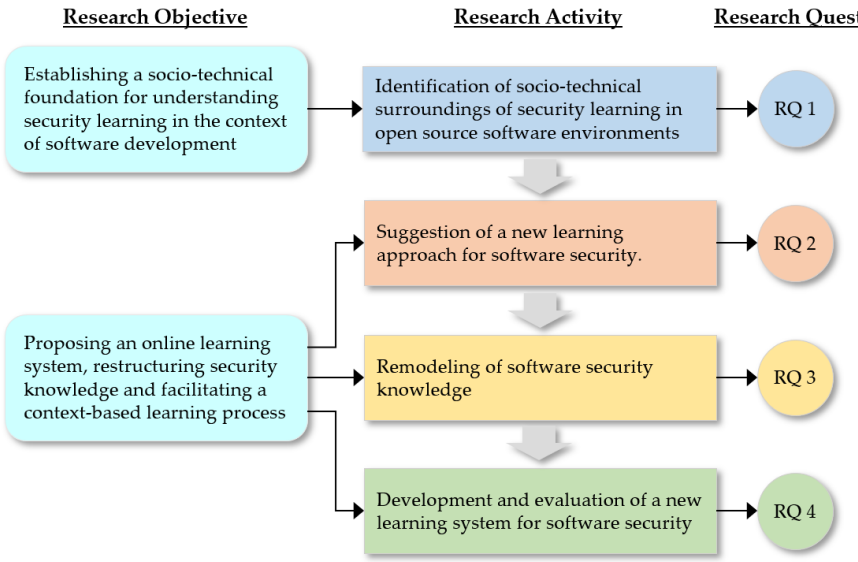


Figure 1.4: Research flow and research questions

would not be viable—nor would it be fair to readers to present background information of seemingly trivial importance. Since the notion of software development has evolved from a different context, it is essential to investigate the research topics within the field in which the software security learning process is embedded and implemented. In this case, the context of open-source software development was chosen. OSS has had a growing impact on society and today's ICT systems: approximately 80% of companies run their operations on OSS [330], and 96% of applications utilize OSS as software components [50]. In 2018, the Linux Foundation[16] reported that the Linux kernel has been committed over 25 million lines of code from over 33,000 open source contributors [261]. However, over 80% of OSS project maintainers and users believe developers should own security, but they aren't well-equipped, according to the State of Open Source Security Report - 2019 [420].

Research question 1 was elaborated into more detailed research questions to establish the magnitude of real-world problems in OSS development.

RQ 1-1: What are the strengths and weaknesses, both technical and non-technical, of software security research conducted in the setting of OSS development?

Many studies have been conducted by both researchers and practitioners on the practices of building security into OSS applications. This research question untangles the domain by investigating the research challenges related to OSS security practices

[16] The Linux Foundation (LF) is a non-profit technology consortium founded in 2000 as a merger between Open Source Development Labs and the Free Standards Group to standardize Linux.

in the literature, and it aims to discover gaps in current research and to thus define relevant research opportunities. This research question is answered in the research paper I (RP I; listed in section 1.5).

RQ 1-2: What issues and challenges need to be addressed and managed to develop and maintain sound security culture in the OSS development context?

Organizational cultures lead people to behave and interact in certain ways, which can be either helpful or harmful regarding learning and job satisfaction [393]. Specific elements of an organization's culture may affect the organization's capacity to learn and may influence what it learns and how it does so [281]. This research question aims at (a) framing the key social and cultural dimensions of software security in OSS development and (b) investigating the current state of security maturity in OSS development through a security culture assessment. This research question is answered in RP II.

RQ 1-3. How have technical, cultural and social aspects affected software-security learning in OSS development?

Open-source software is developed collectively by the online community of practices with a strong relationship between technical and social interactions in a knowledge-intensive process [198, 245]. Therefore, we must recognize and value the setting as a social, spatial, and temporal framework within which learning occurs in the interplay between social and technical aspects. Many OSS proponents believe that OSS development offers significant learning opportunities based on its best practices [204, 257], which are different from traditional educational models [71, 144]. However, studies specifically exploring security knowledge learning in OSS development are quite rare. Hence, this research question involves identifying socio-technical factors in OSS development that influence security learning and investigating structural dependencies among them. The answer to this research question is outlined in RPs III and IV.

RQ 2: How can context-based approaches be applied in software security to motivate learners and to improve learning outcomes?

The traditional security instruction design does not effectively draw learners' attention and is not particularly successful at fostering effective learning of security knowledge. Context-based teaching and learning approaches, however, have been demonstrated in various scientific teaching and learning environments. Yet, it remains unclear how this concept can be synthesized in the domain of software security and how to apply it in the construction of learning materials. This question investigates how security learning can be facilitated via a context-based approach and to what extent this approach motivates students' learning of software security in terms of knowledge gain and learning satisfaction. Research question 2 is split into two sub-research questions, both answered in RP V.

RQ 2-1: What is the design of a learning approach to software security that considers real software scenarios integrated with corresponding security knowledge?

Context-based learning usually takes the form of real-world examples of problems that help to sequence the delivery of facts and concepts; it hence creates a mental model for orienting oneself toward the learning subject. This research question focuses on designing a context-based approach to software security learning that adapts these strategies to software security teaching and learning.

RQ 2-2: What effect does the proposed context-based learning approach have on students' learning outcomes and learning satisfaction?

Building on RQ 2-1, RQ 2-2 is based on the premise that to improve the effectiveness of security learning, the learning approach must promote positive learning outcomes and learning satisfaction. This question investigates whether the proposed learning approach more effectively supports students in learning about software security than traditional methods. This research question also assesses the potential of the proposed learning approach to guide learning material construction.

RQ 3: How can one design an ontology that manages contextualized software security knowledge?

To address weaknesses in security learning regarding knowledge management, including information overload and isolated security concepts, this thesis remodels security knowledge so that it can be retrieved in a manner that takes real-world cases into consideration. Ontologies make this kind of goal possible since they facilitate the capture and construction of domain knowledge and enable the representation of skeletal knowledge [181]. To answer this research question, the thesis first addresses the design pattern of an ontology for appropriately managing contextualized and theoretical security knowledge. Next, it applies ontology evaluation techniques to assess the ontological artifact in terms of its feasibility and applicability in constructing an ontology-based learning system. This research question is answered in RP VI.

RQ 4: How can one construct a learning system that facilitates context-based learning of security knowledge in software development?

While RQ 2 and RQ 3 investigate the feasibility and effectiveness of the proposed context-based learning approach and the ontological knowledge base, respectively, RQ 4 focuses on integrating the two artifacts into the development of a learning system, and it is divided into three sub-questions.

RQ 4-1: How can the proposed context-based learning approach and ontology be appropriately integrated into a contextualized learning system?

This sub-research question investigates how to develop a web-based learning system for software security, which utilizes developed ontology as the kernel knowledge base, meanwhile, facilitates the contextual learning process following the proposed learning approach. The answer to this research question is given in the research paper RP VII.

RQ 4-2: What are the effects of the learning system on students' learning of software security in terms of learning outcomes and learning satisfaction?

The second sub-research question was answered via a preliminary evaluation of the learning system in the context of a controlled laboratory experiment. The aim was to validate whether the system has a positive effect on learning performance and whether it can stimulate learners' interest. This research question is answered in RP VIII.

RQ 4-3: To what extent does the proposed security learning system affect the learning outcome in OSS development environments?

After the initial validation in the school context, the next step was to evaluate the security learning system in a real-world setting, namely, the OSS development environment. To measure learners' satisfaction, this research question explores the perceived usability of OSS developers in terms of system features and the embedded learning approach. The answer to this research question is in RP IX.

1.5 List of Included Publications

Because software development is a field of applied research that draws upon different research disciplines, such integrative efforts are important for identifying important research contributions in each discipline [478] and subsequently the advancement of software development excellence. This research was conducted within a multi-disciplinary academic framework at Norwegian University of Science and Technology, which resulted in a number of research papers (RP) on different disciplines, including sociology, education, information system and others that give important insights to software security learning. This section provides a list of the nine research papers included as part of this thesis, published in either international journals or international conference proceedings. Figure 1.5 illustrates the relationship between research questions and the included research papers. The extended descriptions of the linkages (research questions, research studies, research papers, and contributions) will be presented in Chapter 5.

With the goals of investigating contextual factors that affect developers' learning of software security and suggesting context-based artifacts for effective security education and learning, this thesis contributes to the fields of software development and security education. In Figure 1.6, an overview of the contribution of research papers to academic disciplines is presented, which is placed on a continuum of social

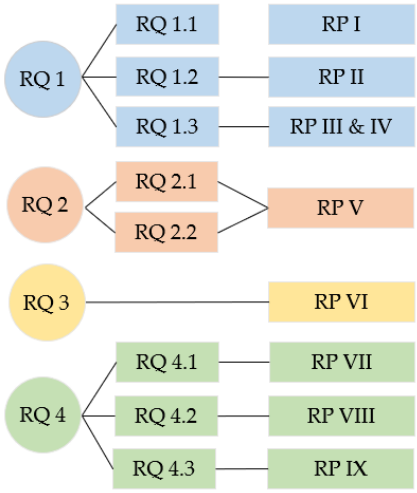


Figure 1.5: The relationship between the research questions and research papers

and technical disciplines with sociology represented at one end and information technology on the other. The length of the bar graphs represents the amount of study that was undertaken.

1. **RP I [481]:**

   Wen, Shao-Fang. "Software security in open source development: A systematic literature review." In *2017 21st Conference of Open Innovations Association (FRUCT)*, IEEE, 2017, pp. 364-373. doi: 10.23919/FRUCT.2017.8250205.

   Academic discipline: Information System, Software Engineering

2. **RP II [490]:**

   Wen, Shao-Fang, Mazaher Kianpour, and Stewart Kowalski. "An Empirical Study of Security Culture in Open Source Software Communities." *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2019, pp. 863-870. doi: 10.1145/3341161.3343520

   Academic discipline: Sociology, Information system.

3. **RP III [483]:**

   Wen, Shao-Fang. "Learning secure programming in open source software communities: a socio-technical view." In *Proceedings of the 6th International Conference on Information and Education Technology*, ACM 2018, pp. 25-32. doi: 10.1145/3178158.3178202.

   Academic discipline: Sociology, Information system.

4. **RP IV [482]:**

   Wen, Shao-Fang. "An Empirical Study on Security Knowledge Sharing and Learning in Open Source Software Communities." *Computers*, 2018, volume 7, issue 4. doi: 10.3390/computers7040049.

   Academic discipline: Sociology, Information system.

5. **RP V [489]:**

   Wen, Shao-Fang and Katt, Basel. "Towards a Context-Based Approach for Software Security Learning." *Journal of Applied Security Research.* 2019, volume 14, issue 3, pp. 288-307. doi: 10.1080/19361610.2019.1585704.

   Academic discipline: Education, Information System

6. **RP VI [486]:**

   Wen, Shao-Fang and Katt, Basel. "Managing Software Security Knowledge in Context: An Ontology-Based Approach." *Information* 2018, volume 10, issue 6. doi: 10.3390/info10060216.

   Academic discipline: Information System, Information Technology

7. **RP VII [484]:**

   Wen, Shao-Fang and Katt, Basel. "Development of Ontology-Based Software Security Learning System with Contextualized Learning Approaches." *Journal of Advances in Information Technology.* 2019, volume 10, no. 3, pp 81-90. doi: 10.12720/jait.10.3.81-90.

   Academic discipline: Information Technology

8. **RP VIII [487]:**

   Wen, Shao-Fang and Katt, Basel. "Preliminary Evaluation of an Ontology-Based Contextualized Learning System for Software Security." In *Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2019, pp.90-99. doi: 10.1145/3319008.3319017.

   Academic discipline: Software Engineering, Education

9. **RP IX [485]:**

   Wen, Shao-Fang and Katt, Basel. "Learning Software Security in Context: An Evaluation in Open Source Software Development Environment." In *Proceedings of the 14th International Conference on Availability, Reliability, and Security*. ACM, 2019, pp 58-67. doi: 10.1145/3339252.3340336.

   Academic discipline: Software Engineering, Information System

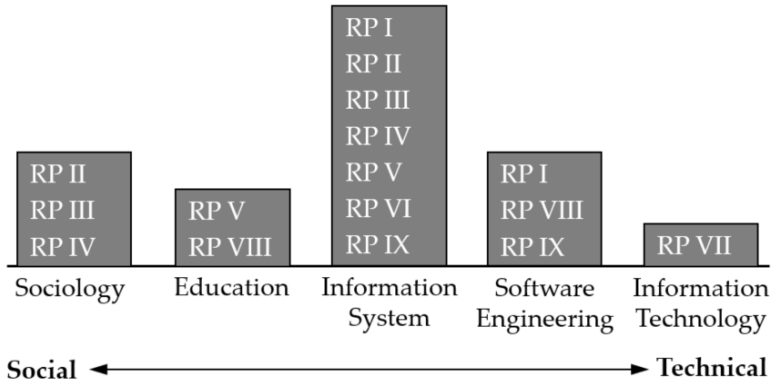


Figure 1.6: Contribution of research papers to academic disciplines

1.6 Thesis Structure

This thesis is comprised of fifteen chapters that are divided into two parts. Part I of the thesis presents an overview of the research work and Part II presents the included research papers.

Part I: Introductory Chapters

Chapter 1: (present chapter) presents an overview of the thesis and consists of sections on research context, problem description, motivation, research objectives, research questions and the list of publications.

Chapter 2: presents a comprehensive and necessary scientific foundation and related work of the research subject areas. The theoretical and practical underlying topics are discussed. The topics include fundamentals of software security, ontology modeling for secure software knowledge, context-based learning perspectives, the theory of the socio-technical system and open source software development.

Chapter 3: presents the complete theorizing process and methodological aspects underpinning the research. It describes the overall research design and explains how theoretical and empirical work has been combined.

Chapter 4: presents an extended summary of the included papers published in peer-reviewed internationally recognized conferences and journals. Each paper presented followed an IMR format: **I**ntroduction, **M**ethodology, and **R**esult. Full research papers are provided in Part II of this thesis.

Chapter 5: highlights and reflects upon the main contributions of this research.

Chapter 6: presents the conclusion of the research work, which includes limitations of the research that are mentioned, followed by some future research opportunities.

Part II: Published Research Papers

Chapters 7-15 include the nine research papers that constitute the main part of this thesis. The papers are presented in the same sequence as in Section 1.5.

Chapter 2

Scientific Background and Related Work

This chapter is divided into six sections. Section 2.1 presents an overview of software security, including basic concepts, terms, secure software development and knowledge for software security. Section 2.2 discusses the teaching and learning of software security, including the teaching approaches, conventional learning materials and tool-based learning for software security. Context-based learning aspects are introduced in Section 2.3 while ontology modeling is presented in Section 2.4. Section 2.5 is devoted to the theory of the socio-technical system, followed by an overview of open source software development, including OSS security and learning in OSS communities, presented in Section 2.6.

2.1 Fundamentals of Software Security

2.1.1 Concepts of Software Security

The field Software Security made its first formal appearance in books and academic classes in 2001 [293]. Software Security is defined as the idea of engineering software so that it continues to function correctly under malicious attack [294]. It is about building secure software: designing software to be secure and making sure that software is secure [293]. The objectives of software security are the preservation of security properties, including confidentiality, integrity, and availability (CIA) [516]; and accountability if their preservation fails. Confidentiality, preventing unauthorized disclosure, and integrity, preventing unauthorized alteration, require mechanisms to firmly establish identities – authentication – and to allow only authorized actions – e.g., access control. Preserving availability includes preventing unauthorized destruction and ensuring adequate access or service. Accountability

includes the ability to later reestablish the acts that occurred and their related actors and ensuring relevant actors are unable to deny an act occurred – non-repudiation. Software Security is an emergent, system-wide property of a software system that takes into account various aspects along software development lifecycle (SDLC), including designing software to be secure, making sure that software is secure, and educating software developers and architects, and subjecting all software artifacts to thorough objective risk analyses and testing [294, 359, 378].

Security studies indicate that most software security problems arise from bugs and flaws during the development process [173, 503]. For example, some of these defects are caused by design and coding issues such as inadequate authentication, improper neutralization of user input, or failure to protect data. This means that one cannot presume to achieve a high level of security by simply introducing security-related features into the software [294, 298]. Security features (as known as Application Security [293]) such as sandboxing code (as the Java virtual machine does), password encryption, and SSL (Secure Socket Layer) between the web server and a browser are functions of an application to prevent malicious attacks. As Michael Howard, a program manager on Microsoft Security Engineering Teams, says "Security features != Secure features" [210]. Security features are used to protected software and the systems that software runs in a post facto way after development is complete. Software security, on the other hand, is more than just security features, which aims to avoid security errors in software by considering security aspects throughout the whole SDLC. It is important to understand that there is no way to guarantee that software is 100% secured. The main idea behind Software Security is to integrate the more level of security possible in software to diminish the possibilities of an attack [513].

2.1.2 Terminologies of Software Security

Terminology is the discipline concerned with the formation, description, and naming of concepts in specialized fields of knowledge, which is a key component in the general documentation process and knowledge formalization [419]. A lot of terminologies used in software security has not been standardized [32]. This section outlines the major terms in the domain of software security that were commonly found in the literature.

Coding Error often refers to bugs in a software program, which causes it to operate incorrectly [206]. These bugs made by the developers in the implementation stage of SDLC that leads to represent the design decision incorrectly in the source code. There are three basic categories of coding errors: (1) syntax errors, (2) runtime errors, and (3) logical errors. In the first two cases when an error occurs, the computer displays an 'Error Message', which describes the error, and its cause. Unfortunately, error messages are often difficult to interpret and are sometimes misleading. In the final case, the program will not show an error message, but it will not do what the

programmer wanted it to do. Coding errors can exist only in code, and may never be executed.

Design Flaw is also a software problem, but a flaw is a problem at a deeper level. Flaws are often much more subtle than simply an off-by-one error in an array reference or the use of a dangerous system call. A flaw is instantiated in software code but is presented at the design level. Design flaws can also be referred to as architectural bad smells [157], design pattern defect [312] or architectural flaw [308]. A design flaw can affect the quality properties of the system and can be caused by an incorrect implementation (or omission) of a design pattern or failure to apply design principles properly [206].

Software Weakness is "a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC" [104]. It refers to issues in software development that may have a direct or indirect impact on software security. Coding errors and design flaws are included in software weakness. For example, if a program routine does not perform input validation, then it 'might' permit unintended or unauthorized behavior. Therefore, a weakness identifies patterns or behaviors that could contribute to unintended behavior. When the weakness can be used by an attacker against the software or another user, then it is a vulnerability.

Software Vulnerability is "an occurrence of weakness (or multiple weaknesses) within the software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness" [103]. Charles P. Pfleeger [354] defines software vulnerability as "a weakness in the security system, for example, in procedures, design, or implementation that might be exploited to cause loss or harm". Software vulnerabilities constitute a majority of security problems, which make software systems open to exploitation and attacks. Some of the common software vulnerabilities include buffer overflow, format string vulnerability, Cross-Site Scripting, Cross-Site Request Forgery (CSRF) and SQL Injection, etc. These 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 [354].

Exploit is a piece of software, a chunk of data, or a sequence of commands that takes advantage of vulnerabilities in an operating system, applications or any other software code, including application plug-in or software libraries to cause unintended or unanticipated behavior to occur on computer software [206]. Such behavior frequently includes things like gaining control or computers, steal network data, allowing privilege escalation, or denial-of-service (DoS) attack. While being used as a verb, exploit refers to the act of successfully making such an attack. In some cases, an exploit can be used as part of a multi-component attack. Instead of using a

malicious file, the exploit may instead drop another malware, which can include backdoor Trojans and spyware.

Attack Pattern is a description of the common attributes and approaches employed by adversaries to exploit known weaknesses in cyber-enabled capabilities [70]. Attack patterns define the challenges that an adversary may face and how they go about solving it. They derive from the concept of design patterns applied in a destructive rather than constructive context and are generated from in-depth analysis of specific real-world exploit examples. They also provide, either physically or in reference, the common solution pattern for preventing the attack. Such a practice can be termed defensive coding patterns.

Security Risk encompasses the probability of occurrence for uncertain events and their potential for loss on software security [234, 458]. An important part of dealing with risk is the method of risk management. Risk management has two distinct flavors in software security. The term risk analysis to refer to the activity of identifying and ranking risks at some particular stage in the software development lifecycle [294]. Risk analysis is particularly popular when applied to architecture and design-level artifacts. On the other hand, the term risk management to describe the activity of performing a quantity of discrete risk analysis exercises, tracking risks throughout development, and strategically mitigating risks.

Secure Coding or secure programming is a set of practices that applies security considerations to develop computer software in a way that defenses against the accidental introduction of security vulnerabilities [459]. In most cases, it implies a programming style that bears security implications of code and implements a defensive code that resists malicious exploits. Secure coding standards introduce safeguards that reduce or eliminate the risk of leaving security vulnerabilities in code. For applications to be designed and implemented with proper security requirements, secure coding practices and a focus on security risks must be integrated into day-to-day operations and the development processes.

2.1.3 Secure Software Development

To achieve software security, developers need to build assured software; “Software that has been designed, developed, analyzed and tested using processes, tools, and techniques that establish a level of confidence in its trustworthiness appropriate for its intended use” [392]. To achieve this goal, developers must rethink the software development process and address security in all the phases of the SDLC: definition of the requirements, architecture and design, coding, testing, validation and maintenance of the software [211, 296, 421, 459]. This is like applying the defense-in-depth strategy to the various phases of the software development lifecycle making it more security-aware [210]. The use of Secure Software Development Lifecycle (SSDLC) is of utmost importance if the objective is not only the prevention of security bugs but also higher-level problems, like architectural, component interaction and

broken access control over tiers. This way of developing secure applications has already proven results from the industry taking into account both secure mechanisms and design for security [149, 211]. Following are the five major activities of SSDLC in general:

1. Security Requirement: Since the idea of secure development is to start at the very beginning of the coding, security must be grounded in system requirements and specify system functions in all possible circumstances of use, legitimate or malicious.

2. Security Architecture and Design: Considering security and privacy in the initial design of new products and features permits the integration of security in a way that minimizes disruptions to plans and schedules. Architecture risk analysis, which is referred to as threat modeling, can be used to prevent and detect design flaws.

3. Secure Coding: To avoid coding issues that could lead to vulnerabilities and leverages state-of-the-art development tools to assist in building more secure code. Analyzing the source code (static analysis tools) before compile provides a scalable method of security code review and helps ensure that secure coding policies are being followed.

4. Security Verification and Testing: Executing run-time verification of software applications to ensure that functionality works as designed. Apply appropriate verification to software applications and make sure they produce proper functionality as defined in the initial design.

5. Release and Operation: Preparing response plans and protocols to address new threats that emerge over time. Certifying software before a release helps ensure security and privacy requirements were met.

To address security activities explicitly in the software development process, many SSDLC models or frameworks have been proposed to embed security practices along SDLC. For example:

- Microsoft Security Development Lifecycle [211] — MS SDL— is one of the first of its kind, the MS SDL was proposed by Microsoft in association with the phases of a classic SDLC.

- NIST 800-64 [238] provides security considerations within the SDLC. Standards were developed by the National Institute of Standards and Technology to be observed by US federal agencies.

- OWASP Comprehensive, Lightweight Application Security Process [343] — CLASP — is an activity-driven, role-based set of process components guided by formalized best practices. CLASP is designed to help software development teams build security into the early stages of existing and new-

start software development life cycles in a structured, repeatable, and measurable way.

- Software Security Touchpoints [294]. Gary McGraw provided seven software security touchpoints by codifying extensive industrial experience with building secure products. McGraw uses the term touchpoint to refer to software security best practices which can be incorporated into a secure software lifecycle.

2.1.4 Knowledge for Software Security

Knowledge is more than simply a list of things we know or a collection of facts [62]. "Knowledge is information in context."[62]. Knowledge is the accumulation of information; information made actionable, knowing and understanding how to apply gained information to perform tasks [9, 62, 512]. A checklist of secure coding practices for web applications is information; the same information that developers understand the whys and wherefores of it in a software project is knowledge. Similarly, a set of results generated from the static analysis tool is information; to analyze the meaning and take actions need knowledge. In software engineering, security knowledge is multifaceted and can be applied in diverse ways [294]. It provides a foundation that can be directly or dynamically applied through knowledge-intense practices along the SSDLC [31]. Figure 2.1 depicts that software security knowledge plays a central role in supporting all security activities along SSDLC. Designers, programmers, and testers need to be aware of possible security errors, potential attacks and relevant countermeasures that minimize the exposure to security problems [496].

Secure software knowledge falls naturally into three categories: the nature of attacks, how to defend, and the computing system's environment in which the conflict takes place [370]. It is comprised of domain knowledge in software security and situated

Figure 2.1: Security knowledge and secure software development lifecycle

knowledge grounded in the developers' unique software development environment. In general, domain knowledge is the fundamental knowledge obtained through long and deliberate learning [129]. It includes theoretical knowledge that the expert acquires through formal education, training or certification [79]. Situated knowledge is dynamic and organization dependent. This type of knowledge is hard to articulate, and the developers acquire it through continued interactions with a specific operating environment. For example, the security principle of least privilege recommends that accounts should have the least amount of privilege required to perform the task. This encompasses the security practices of user rights, and resource permission such as CPU, memory, and network, which exist for specific programming languages (e.g. C, C++, PHP, Java and so on), depending on the features of the software product.

For reasons of clarity and ease of application of security knowledge, Barnum and McGraw [31] proposed a knowledge schema for software security. Figure 2.2 shows the knowledge catalogs (the boxes) and their relationships. In their model, seven knowledge catalogs (principles, guidelines, rules, vulnerabilities, exploits, attack patterns, and historical risks) are grouped into three knowledge categories (prescriptive knowledge, diagnostic knowledge, and historical knowledge). The prescriptive knowledge category includes actions or procedures which offer advice for what to do and what to avoid when building secure software, like security principles, guidelines, and rules. Rather than prescriptive statements of practice, diagnostic knowledge helps practitioners (including operations people) recognize and deal with common problems that lead to security attacks. Attacks, exploits, and vulnerabilities are therefore classified as diagnostic knowledge. Historical knowledge helps the practitioner to understand the real problem based on extensive experience with the same or a similar problem. This catalog represents detailed descriptions of specific issues uncovered in real-world software development efforts and must include a statement of impact on the business or mission proposition. Common security problems like vulnerabilities and corresponding attacks can be detected and dealt with using prior experience with these problems.

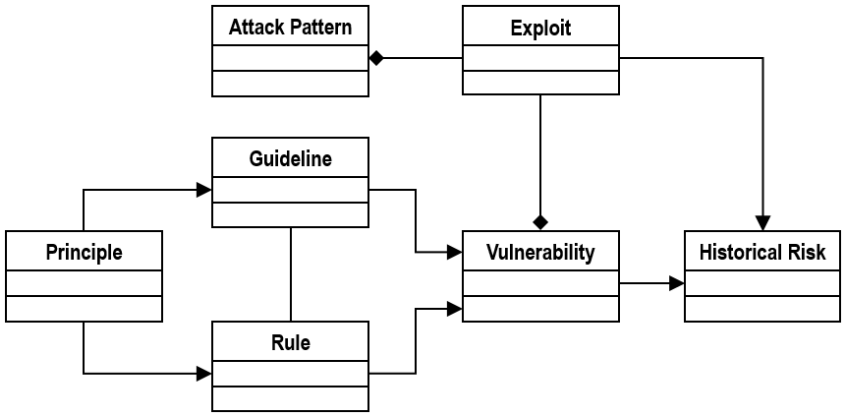


Figure 2.2: The software security knowledge schema proposed by Barnum and McGraw [31]

Base on the security knowledge schema presented above, the main knowledge resources for the catalogs (Attack, Vulnerability, Principle, Guideline, and Rule) are presented below:

Attack Pattern. The Common Attack Pattern Enumeration and Classification (CAPEC)[17] system provides a publicly available catalog of common attack patterns that helps users understand how adversaries exploit weaknesses in applications and other cyber-enabled capabilities. provides a formal list of known attack patterns. In their research paper, Sean and Amit [32] introduce the concept, generation, and usage of attack patterns as a valuable knowledge tool in the design, development, and deployment of secure software. McGraw teamed with Greg Hoglund for the book *Exploiting Software: How to Break Code* [206], which offered a taxonomy and discussion of security attack patterns.

Vulnerability. The CVE system provides a reference-method for publicly known software-related vulnerabilities and exposures. CVE's common identifiers make it easier to share data across separate network security databases and tools and provide a baseline for evaluating the coverage of an organization's security tools. The Common Weaknesses Enumeration (CWE)[18] is a category system for software weaknesses and vulnerabilities. It is sustained by a community project with the goals of understanding flaws in software and creating automated tools that can be used to identify, fix, and prevent those flaws.

Principle. Some research groups proposed principles for secure software development [173, 231, 389, 427, 460]. OWASP has provided a comprehensive list of security design principles that programmers should adhere to [344].

Guideline. OWASP provides a general secure coding reference guide in a checklist format that users can integrate into the development life cycle [342]. Software Assurance Forum for Excellence in Code (SAFECode)[19] has developed a guide outlining fundamental practices for secure software development [386]. Some secure coding guidelines are provided by technology vendors, for example, Oracle Corporation [338], Apple Inc. [22], and Mozilla [316].

Rule. The CERT Coordination Center (CERT/CC)[20] has released detailed coding rules for several common programming languages (e.g., C, C++, and Java) [403]. Motor

[17] https://capec.mitre.org/
[18] https://cwe.mitre.org.
[19] The Software Assurance Forum for Excellence in Code (SAFECode) is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance method: https://safecode.org/
[20] CERT/CC is a non-profit United States federally funded research and development center: https://www.sei.cmu.edu/about/divisions/cert/

Industry Software Reliability Association (MISRA)[21] provides a set of software development standards and rules in C/C++ [310], which enable best practices in code safety, security, portability, and reliability in embedded systems.

In addition to these knowledge resources, several research groups focused on building Common Body of Knowledge (CBK) to provide a comprehensive framework of all relevant subjects that a security professional should be familiar with, including skills, techniques and best practices. The United States Department of Homeland Security has developed a common body of knowledge for software assurance (SwACBK) [370]. The Cyber Security Body of Knowledge project (CyBoK) [356] funded by the U.K. National Cyber Security Program aims to inform and underpin education and professional training for the cybersecurity sector.

To formalize security knowledge, be following the design consideration of the contextualized learning system, this thesis first considered which terms are critical in explaining software security knowledge without overlapping between concepts they represent, further, extracted corresponding knowledge from the available resources to construct the security knowledge, as the kernel of the learning system. The corresponding research work is described in RP V and RP VI.

2.2 Teaching and Learning Software Security

2.2.1 Approaches for Teaching Software Security

Education is an essential tool to help produce secure code in software engineering [321]. To ensure all software engineering graduates have the knowledge necessary to develop secure software systems, security experts and educators emphasize education must infuse security principles and secure programming early and often in the learning process [48, 209]. As such, a number of researchers are investigating various methods for integrating security practices into the computer science and information system management curriculum. Perrone et al. [353] and Taylor and Azadegan [435] recommend a threaded approach intended to reach all students in the computer science and information system curriculum. Security principles and secure coding practices are interleaved into the curriculum, starting with the foundation courses and re-enforced throughout the student's course of study. Similarly, Chung et al. [82] proposed to develop a secure software engineering-based thread approach. In their proposal, students develop software with software-engineering case studies, then they are demonstrated how the produced code can be transformed to include security across the life cycle, resulting in secured code. Kara Nance [320] presented projects in introductory classes that asked students to deal with security problems such as file recovery and printer forensics. These require

[21] MISRA is a collaboration between manufacturers, component suppliers and engineering consultancies which seeks to promote best practice in developing safety- and security-related electronic systems and other software-intensive application: https://www.misra.org.uk/

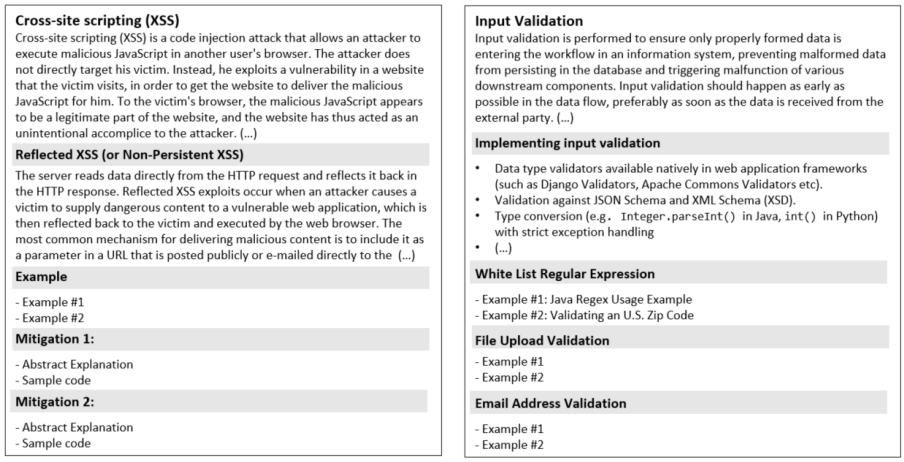
integrating security material into the class curriculum. Bishop et al. [46, 111] then proposed the use of Secure Programming Clinic (SPC) to provide practical educational training for students that extends and reinforces the theory they learn in classes. In SPC, students send their code to the clinicians, usually manned by graduate students, who review the code and then have one-to-one discussions with students about the security implication of their code.

The above security integration teaching techniques have been recognized as pedagogically effective, which have an advantage in that security concepts can be transferred more easily across both core and elective curricula, as well as different kinds of institutions [82, 435]. However, adopting this approach, institutions need substantial time and funds to upgrade curriculum to include security topics (either in the curriculum itself or in faculty time to develop new course materials or alter existing ones), as well as to prepare a sufficient number of skilled resources, including trained faculty, to support the program. We advocate to further explore the context-based approach to complement the security integration approach, without causing the severe expense of depth and breadth of a curriculum change for resource-limited institutions. We contribute to this approach by concretely exploring a viable implementation solution and evaluating its effectiveness.

2.2.2 Conventional Security Learning Materials

In the learning process, learning materials is one of the main factors to be considered by the instructor because it can contribute to the acceptance of students of knowledge presented. Learning material can consist of various forms and formats depending on the teaching methods. When most institutions plan and develop security learning materials, either textbooks, lectures or online courses, they commonly use conventional approaches to guide the process. Such conventional learning materials and approaches are commonly made up of two distinct methods: black-hat concepts and white-hat concepts (offense/defense, construction/destruction) [294]. Figure 2.3 illustrates the two types of learning materials. The black hat/white hat concepts apply the classic western “bad guy/good guy” concept to software security. A black-hat refers to a hacker who tries to break into a system with malicious intent. Black-hat actions include destructive activities such as attacks and exploits [294]. Using a black-hat approach in software security implies thinking proactively about ways that a system could be exploited. A white-hat refers to an individual who identifies a vulnerability in a system and reports it to the system owners. White-hat actions include constructive activities such as design, defense, and functionality [294]. Using a white-hat approach in software development includes building defense into a system, often using information from a black-hat history.

The conventional learning materials typically address particular security topics, and the starting point of instructions consists of basic security concepts and theories, which are taught in a logical order and structure. In addition, these learning materials are often written in the form of a reference manual or a guide to a particular security



(a) (b)

Figure 2.3: Two types of conventional learning materials for software security: (a) the black-hat approach, and (b) the white-hat approach

certification, which is more effective at training security experts. However, it is difficult for developers to correlate what they are learning to their programming experience, further, to link the security knowledge to real software scenarios. In this approach, the interests and thoughts of developers and the knowledge they already possess are not taken into account, which could lead to forced concept development and misconceptions. An ideal learning process should therefore also be guided by the motives, skills, and pre-knowledge of students. Since security learners have to demonstrate the applicability of the knowledge through experience in order to understand their practical use [294], the learning materials presented must provide meaning for learners, allowing them to learn security principles closed to the real-world situations that are of particular interest to them.

2.2.3 Tool-Based Support for Learning Software Security

Some efforts have been made to enhance software security education and learning using tool-based learning approaches. In this section, various types of security education tools from the literature are briefly introduced.

Atsuo Hazeyama et al. [195, 196] proposed an artifact-driven learning process for software security as well as an online learning environment utilizing a body of knowledge for security education. In the learning process, learners conduct secure software development by inputting artifacts that were created in a traditional software engineering course, such as requirements specification, use case diagram, and test specification. The learning flow takes security into consideration after considering the functional requirements of a system. The designed learning environment provides functionalities for maintaining artifacts that are inputs for

learning about software security and outputs from that learning, furthermore, giving relationships between artifacts and the reference information. Learners proceed to learn about software security by referring to the available information.

In addition to online programs, the Integrated Development Environment (IDE) plug-in has been applied in teaching students or programmers about security awareness. The University of North Carolina at Charlotte (UNCC) has designed and developed an Educational Security in the Integrated Development Environment (ESIDE) plug-in for Eclipse[22] that delivers real-time secure programming instructional support as students write code [492, 517], similar to the underline in a word processing spell checker. The tool is designed to improve student awareness and understanding of security vulnerabilities and to increase the utilization of secure programming techniques in assignments. ESIDE aims to provide educational interventions for more advanced students (a senior and masters-level web development course) [492], and the tool only works on Eclipse IDE for Java and cannot support other platforms, for example, the Android IDE.

Visualization is another approach in teaching software security courses, which heavily uses images, diagrams or animations to communicate messages [401]. To integrate visualization techniques in classroom instruction, Yuan [509] developed three visualization and animation tools that demonstrate various information security concepts. The information security concepts illustrated include packet sniffer and related computer network concepts, the Kerberos authentication architecture, and wireless network attacks, through the usage of Macromedia's Flash software. Bishop et al. [47] have developed a Concept Map[23] of secure programming to visualize the relevant body of knowledge, which assists students in understanding complex concepts, principles, and ideas and the important relationships between them. Their concept maps are assessments designed to identify students' misconceptions; the questions, scoring procedures, and interpretations are consistent and in adherence with a predetermined standard. The results from the concept map are primarily intended to improve pedagogy, though the results can be used to help instructors make comparisons of teaching over time.

Furthermore, video games (game-based learning), such as CyberCIEGE [216] and hACMEgane [324], are approaches taken to stem the declining interest and enrollment in computing courses, where students explore relevant security aspect of games in a learning context designed by the instructor. CyberCIEGE[24] is a free tool that can be downloaded from the Internet for that purpose. Students can build their networking environment virtually and learn the possible threats that affect their network based on their design. Through available security scenarios, students will learn security through the consequences of their choice while they build their own

[22] Eclipse is an integrated development environment used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment: https://www.eclipse.org
[23] http://spc.cs.ucdavis.edu/index.php/conceptmap
[24] https://my.nps.edu/web/c3o/cyberciege

network. In hACMEgame, games are organized as a series of levels where the player must overcome a set of challenges in order to unlock access to the next level. Each level focuses on a set of well-known security vulnerabilities.

The web-based security learning tool proposed in this thesis differentiates from the previous works in two main aspects[25]:

(1) The tool is context-based, in which context-based learning is facilitated in the learning process.

(2) The tool is ontology-based, in which the security knowledge is modeled with contextual situations and incorporating theoretical knowledge to complement the concrete description.

2.3 A Context-Based Learning Perspective

2.3.1 Context and Knowledge

The notion *context* stems from Latin *contextus* "connection, coherence". Basically, it refers to all the aspects that are relevant for an understanding of a certain piece of text, be it written or spoken language ("discourse"). According to Oxford Dictionaries[26], context is defined as "The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood." Dey [59] defined context as, ''context is a set of information used to characterize a situation of an entity". An entity is a person, place, or object that is considered relevant to the interaction between a user and the environment. The concrete or ideal field of a sign-meaning unit, which can support the specification of meanings at a given moment in time, is generally referred to as context [452]. Context provides for two essential processes: on the one hand, it supports the particularization of meanings by restricting the cognitive process of meaning construction, and by eliminating ambiguities or concurrent meanings that do not seem to be adequate at a given moment; on the other hand, context also prevents this particularized meaning from being isolated as it brings about coherence with a larger whole [452].

In the real world, context is a complex description of the knowledge shared on physical, historical and other circumstances where actions or events happen. According to Baskerville [12], knowledge is information combined with experience, context interpretation, and reflection. It is a valorous kind of information that is ready to be used in decisions and actions. Contextual information is a crucial component of fully understanding knowledge [58, 219, 242]. Brézillon [58, 242] points out that knowledge comes from a variety of contexts that cannot be accurately understood without context. Nonaka and Konno [328] also noted that knowledge reflects a

[25] The research work of the proposed learning system is published in RP VII, presented in Chapter 13.
[26] https://en.oxforddictionaries.com/definition/context

particular instance, perspective, or intention in accordance with the characteristics of a specific context, which is different from information. The context has the capacity to provide a major meaning to knowledge, promoting a more effective comprehension of a determined situation in the collaborative work [60]. All this knowledge is not part of the actions to execute or the events that occur but will constrain the execution of an action and event interpretation [57]. Without proper contextual description, knowledge can be isolated from other relevant knowledge, resulting in limited or distorted understanding [61, 169]. Infield observations of the usage of an organizational knowledge management system that stores knowledge about UNIX problems, Ackerman [3] found that users chose not to use the solution provided by the system because they could not determine the appropriateness of the solution without knowing the context in which the solution has been applied, such as the size of the UNIX installation and the organizational setting. Addressing this shortcoming requires knowledge built around real-world scenarios that actively engage learners [91, 384].

2.3.2 Context-Based Teaching and Learning

In domain-specific theories of learning and teaching, the importance of context is also underscored. In mathematics and chemistry education a contextual approach of learning is already viable for a relatively long time now [45, 156, 174, 431]. Contextual problems are generally seen as one of the core concepts in the scientific education movement. Gravemeijer [174], for example, explains how context is viewed in realistic mathematics education: "Contextual problems describe situations where a problem is posed. More often this will be an everyday life situation, but not necessarily so; for the more advanced students' mathematics itself will become a context ([174], p. 105)." Apparently, context is seen here not merely as a synonym for a concrete external situation, but it can assume the character of a mental framework as well. If the learning content is explicitly connected to experiences outside the classroom – and thereby situated or contextualized, the links between the learning environment, especially learning tasks, and learner's pre-knowledge will be built. When a pupil, for example, appropriates the notion of an area in the context of a meaningful activity of covering a table with paper, decorating the floor of a dollhouse with a footcloth, etc., this notion of the area will probably be linked up to other meaningful notions, as surface, size, length, unit of measurement, etc. The context then ties different notions and experiences together, as a result of which the notion of the area will be more than just a formula.

Context can increase the information content of natural language utterances and facilitate learning [57, 59]. Psychology and education researchers have demonstrated that when knowledge is learned in a context similar to that in which the skills will actually be needed, the application of the learning to the new context may be more likely [117, 352]. Predmore [360] showed that learning about knowledge content through real-world experience is important because "once [students] can see the real-world relevance of what they're learning, they become interested and motivated."

The book How People Learn [90] also pointed out that motivation is critical for learning, enabling knowledge transfer to occur. If students do not learn the material well in the first place, they cannot possibly transfer it to new situations. As stated in the book "Learners of all ages are more motivated when they can see the usefulness of what they are learning and when they can use that information to do something that has an impact on others" (page 49).

In context-based approaches, contexts are used as a starting point for the design of innovative curricula, with the intention to tackle a couple of problems perceived in conventional science education. Bennett, Lubben [38] offered a definition of a context-based approach to science education: "Context-based approaches are approaches adopted in science teaching where contexts and applications of science are used as the starting point for the development of scientific ideas." The authors reported that context-based science courses motivate students and help them become more positive about science by organizing learning experiences to take into consideration representing real-world situations of the learning subject. When students are more interested and motivated by the experiences they are having in their lessons, their increased engagement may result in improved learning [38]. Besides the focus of this approach on enhancing the interest and attitude towards science education, the use of context also has the purpose to influence the improvement of learning outcomes and an increased understanding of science by students [468].

In computing science education, there is also a broad agreement that teaching units should start from a "real-world" context or phenomenon, aiming to create connections to prior knowledge, increase the relevance of the material to students, or show applications of the intended knowledge, thereby increasing motivation [120, 184]. These contrast with more traditional approaches that cover abstract ideas first, before looking at practical applications. Likewise, in software engineering, studying in one context and then abstracting the knowledge gained for use in a new context is a common way of learning programming that has been observed extensively in both new and experienced programmers [23, 243]. Digital news, newspaper reports, and even crime and other dramas on TV and movies all provide examples of security learning with context-based approaches. These include the Heartbleed vulnerability in OpenSSL[27], WannaCry ransomware attack[28], eBay's data breach[29] , and Heartland Payment System attack[30]. Learning about secure software knowledge, therefore, is not just about knowledge, but about putting knowledge into context in order to apply security practices effectively. Understanding the context in which software will be deployed and used, the risks and threats of its misuse, and the systematic its development, are increasingly recognized as critical to its success [315]. In order to capture and use security knowledge appropriately, it is necessary to first specify

[27] http://heartbleed.com/
[28] https://en.wikipedia.org/wiki/WannaCry_ransomware_attack
[29] https://news.netcraft.com/archives/2017/02/17/hackers-still-exploiting-ebays-stored-xss-vulnerabilities-in-2017.html
[30] https://www.bbc.com/news/technology-23448639

which context information is to be handled. Then, it must be represented in a format that is understandable and acceptable to the individuals. Thus, a context for a software security topic includes the circumstances in which its technical content exists. Therefore, when talking about software security in a given context, the knowledge would not only include the basic principles and processes of software security but also consider how security knowledge is used in one or more particular domains or application areas.

2.4 Ontology Modeling

2.4.1 Ontology

According to Gruber [180], an ontology is "an explicit and formal specification of a conceptualization", that is, a formal description of the relevant concepts and relationships in an area of interest, simplifying and abstracting the view of the world for some purpose [473]. An ontology is basically a graph whose nodes represent the concepts or objects of a domain, and the edges indicate relationships between concepts. Usually, this graph is structured around a hierarchical "backbone" similar to the class/subclass relationship in object-oriented programming. Due to the formalization, it can be represented and to some degree interpreted by machines, and enables the formal analysis of the domain. This allows an automated or computer-aided extraction and aggregation of knowledge from different sources and possibly in different formats [180]. Figure 2.4 depicts an example of a security ontology proposed by Fenz and Ekelhart [143], in which the top-level security concepts and relationships are presented.

Ontologies are now central to many applications such as scientific knowledge portals, information management, and integration systems, electronic commerce and web services. The main areas, in which ontological modeling is applied, include communication and knowledge sharing, logic inference and reasoning, and knowledge base. By analyzing and extending several types of research [331, 446], we

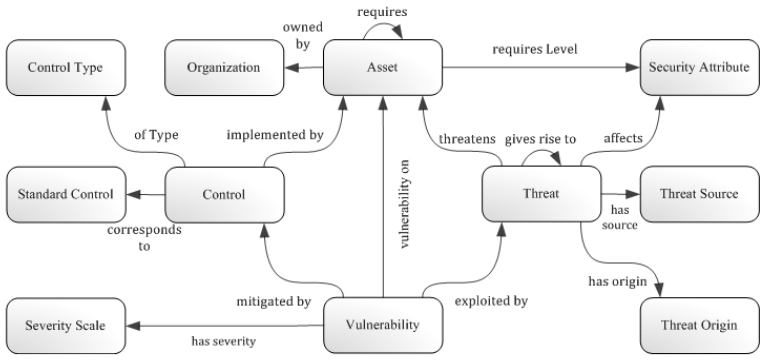


Figure 2.4: Security ontology [143]

can identify and summarize the reasons for and benefits of developing and using ontologies in knowledge modeling.

- Ontologies share a common understanding of structured information among people or software agents.
- Ontologies make domain knowledge reusable.
- Ontologies enable the interoperability among models or specific domain vocabularies.
- Ontologies allow and simplify the communication among humans, computational systems, and between humans and systems.
- Ontologies have the expressive power for acquiring context from the diverse and heterogeneous sources.

2.4.2 Existing Approach

There have been extensive research works in the area of security knowledge modeling and ontology applications to software security. Some papers focus on using an ontology to model security vulnerabilities. Guo and Wang [183] presented an ontology-based approach to model security vulnerabilities listed in CVE. The authors identified critical concepts of security vulnerabilities in the domain of software security, which can be provided for machine-understandable CVE vulnerability knowledge and reusable security vulnerabilities interoperability. Syed and Zhong [432] proposed an ontology-based conceptual model for the formal knowledge representation of the cybersecurity vulnerability domain and intelligence, which integrated cybersecurity vulnerability concepts from several sources including CVE, NVD[31], CVSS[32] framework, and social media. Alqahtani et al. [14] proposed an ontological representation, which established links with bi-directional traceability between traditional software repositories (e.g., issue trackers, version control systems, Q&A repositories) and security vulnerabilities databases (e.g., NVD)

Some researchers presented their ontology in supporting security requirements and design processes in software development. Gyrard et al. [185] proposed STACK ontology (Security Toolbox: Attacks & Countermeasures) that supported developers in secure application design. Countermeasures in STACK included cryptographic concepts (encryption algorithm, key management, digital signature, and hash function), security tools, and security protocols. Kang and Liang [227] presented the security ontology adopting the Model Driven Architecture (MDA) methodology. Their proposed ontology could be used in security concepts modeling in each phase of the development process (e.g., the requirement and design phases) with MDA. In

[31] The National Vulnerability Database (NVD) is the U.S. government repository of standards-based vulnerability management data represented. (https://nvd.nist.gov/)
[32] The Common Vulnerability Scoring System (CVSS) is a free and open industry standard for assessing the severity of computer system security vulnerabilities. (https://www.first.org/cvss/)

order to improve the application of security patterns to the security engineering domain, Guan et al. [182] proposed an ontological approach facilitating security knowledge mapping from security requirements to security patterns. Manzoor et al. [282] developed an ontology, illustrating the relationships across various actors involved in the Cloud ecosystem, to analyze different threats to/from Cloud-system actors.

Finally, some efforts focused on building security ontology specifically in the context of web application development. Salini and Kanmani [388] presented an ontology for defining the security requirements of web applications. The included concepts are assets, vulnerabilities, threats, and stakeholders. Their ontology aimed at reusing the knowledge of security requirements in the development of different kinds of web applications. Busch and Wirsing [65] presented a security ontology for secure web applications (SecWAO), which aimed to support web developers to specify security requirements or make design decisions in web application development. It distinguished various concepts among methods, tools, mechanisms, assets, vulnerabilities, and threats. Velasco et al. [455] presented an ontology-based framework for string, presenting and reusing security requirements. Their framework integrated security standards, methods of risk analysis, and the requirements ontology.

A major feature, which is common for all the above studies, is that the ontologies commonly focus on unifying security concepts and terminologies. Subsequently, they either dedicate to a certain software domain or support part(s) software development processes. The ontology proposed in this thesis differentiates from the previous research work in the following aspects[33]:

(1) The ontology is context-based, which models security knowledge with a diversity of software features and technologies of application contexts;

(2) The ontology describes security knowledge with a contextual situation, and meanwhile, complement the contextualized knowledge with conceptual descriptions.

2.5 Socio-Technical System Theory

2.5.1 Concept of socio-technical system

Socio (of people and society) and technical (of machines and technology) are combined to give 'sociotechnical' (all one word) and/or 'socio-technical' (with a hyphen). Socio-technical refers to the interrelatedness of 'social' and 'technical' aspects of an organization [471]. Socio-technical systems (STS) pertains to a theory regarding the social aspects of people and society and technical aspects of

[33] The research work of the proposed learning system is published in RP VI, presented in Chapter 12.

organizational structure and processes. One of the earliest and most important statements on socio-technical systems and the workplace comes from the English organizational theorists Eric Trist, Ken Bamforth, and Fred Emery at the Tavistock Institute in London, who conducted a study with workers in Britain coal mining industry, in World War II-era [277]. They tried to understand how the social and technical aspects of coal mining worked together as the industry was changing. As a new mining technology was being developed, the industry needed to adapt. The Tavistock socio-technical approach involved recognizing the systems comprised of both technical elements and social elements and that both could be developed in parallel – with benefits both for productivity and quality and for the well-being of the workers. This approach has turned out to be just as important for modern computer-based systems as for the industrialized production systems of the past [493].

The STS in organizational development is an approach to complex organizational work design that recognizes the interaction between people and technology in workplaces. The term also refers to the interaction between society's complex infrastructures and human behavior. In this sense, an STS can be recognized as an *open system* [461], as a way of describing, analyzing and designing systems with joint optimization in mind, particularly those that embody some degree of non-linearity within themselves as well as the environment they reside in. For example, an airplane includes two side-by-side systems with different needs: one technical (the airplane) and one organizational (the pilot with the crew team). In the socio-technical system, these sub-systems must collaborate closely – the pilot must understand the plane's controls, which must also be understandable among the crew. The STS is the airplane plus the pilots and the whole crew team as a single system with human, organizational and technical levels, saying a social system sitting upon a technical base, as physical societies have architectures, that the social contextualizes the technology even as it is created by it.

STS theory was initially developed to improve the quality of working life when workers interact with the technology used by organizational processes. Any ICT or information system is embedded into a social context which adapts to and helps to reshape, social worlds through the course of their design, development, deployment and uses. The strength of STS is that they integrate these different phenomena so that they increase their performance reciprocally. Even more important, the integration of technical and social systems helps them to develop and to constitute each other, for example, the interaction among community members is supported by technical infrastructure, and the members themselves can contribute to the development of the infrastructure, as is typically demonstrated by open-source software communities. The quality, structure and other characteristics of the developed software systems also depend upon the education of software engineers, their work experience, problem-solving strategies, organizational structure, social relations, and shared mental models [108]. Prior ethnographic studies [114, 177, 441] suggest that technical dependencies among software components create "social dependencies" among software developers implementing these components. Therefore, the software

development process is not purely a technical task, but also a socio-technical system embedded within organizational and cultural structures [188].

2.5.2 Applications of STS theories

The STS aspects provide a deeper analysis of the relationship between the methods, techniques, tools, development environments and organizational structures [471]. More recently, there have been efforts to apply the socio-technical system concept to solving problems from software engineering domains. For example, Lu and Jing [278] present a socio-technical approach to support integrated socio-technical negotiation activities in a collaborative software design process. They address the critical issues of such collaborative negotiation activities, including modeling negotiation arguments based on social and technical factors and analyze these arguments to reconcile the conflicts for software design tasks. Ducheneaut [123] examines the socialization of new members in an open-source community using socio-technical analysis since these members interact with both people and material components of a project. Ye et al. [507] propose a socio-technical platform to guide the design of software that supports information seeking and communication during different phases of programming.

In the 1990s, Stewart Kowalski proposed a model of socio-technical systems [250], depicted in Figure 2.5. Through a holistic approach, he modeled the dynamics of technology and social changes that determined the (in)security level of a socio-technical security system. He has argued that every socio-technical system is affected by four components (Culture, Structure, Method, and Machine) belonging to two subsystems (Social and Technical). The "Culture" component refers to the collective and distributed values in actions, while the "Structure" component refers to the abstract authority system. The "Method" component refers to those methods applied to produce work using the existing "Machine". "Culture" and "Structure" belong to the social subsystem, whereas "Method" and "Machine" belong to the technical subsystem. The existing arrows in the model indicate patterns of interchange between the system components. As the system needs to maintain a state of equilibrium, any change happens to one of the system components due to an internal or external factor,

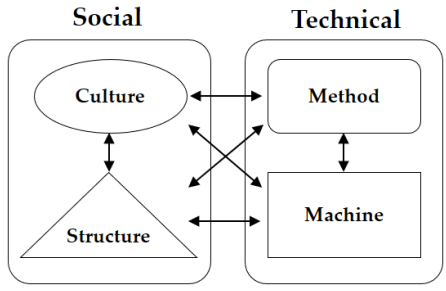


Figure 2.5: A model of Socio-Technical System [250]

the systems other components need to interchange accordingly to keep maintaining the state of equilibrium of the whole system. The STS model has been applied to evaluate threat modeling in the software supply chain [8], business process re-engineering [4], a framework for securing e-Government services [228] an information security maturity model [21]. The STS provides an appropriate and legitimate way to perform system analysis through a systemic–holistic perspective and helps us understand the intrinsic context in the open-source software phenomenon.

2.6 Open Source Software Development

2.6.1 Concepts of open source software

The open source software is a radically revolutionary concept to develop software [368], which began in the mid-90s. In the OSS approach, source code of the software products is made freely made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose [168]. OSS is released under a license conforming to the Open Source Definition (OSD)[34] as articulated by the Open Source Initiative (OSI) [139]. There are different kinds of licenses used in OSS such as Apache License, GNU General Public License (GPL), FreeBSD license, and MIT license. The fundamental idea of OSS is to enable the software to evolve freely by exploiting community participation, making it possible for end-users to adapt the software to their personal needs and fix defects [368]. OSS development model has produced a number of successful applications in the area of operating systems (Linux), emailing and web services (Gmail, Apache), databases (MySQL, PostgreSQL), etc.

2.6.2 OSS development model

The organization of OSS development is fundamentally different from that of traditional project organizations of proprietary ("closed source") software. In contrast to the centralized governance, OSS is an extreme case of geographically distributed software development, free of hierarchical control structures for the establishment of standards, verification, and distribution of a particular software application to build a particular application, so-called the Bazaar model [368]. The OSS development project has a unique socio-technical structure depending on the nature of the system and its member population. In general, the initial OSS developer maintains a lead role and is responsible for the governance and the coordination process [506]. The project leader, or the core team, usually partition the software development tasks into manageable modules and has participants choose what to work on according to their

[34] The Open Source Definition (OSD) is a document published by the Open Source Initiative, to determine whether a software license can be labeled with the open-source certification mark. https://opensource.org/osd

interests. These volunteer workers coordinate their activities through elaborate infrastructure over the Internet, such as the mailing list (for communication between all the interested parties – from users to developers) as well as Concurrent Versions System (CVS)[35]. 'GitHb.com' and 'Sourceforge.net' provide web-based portals for most OSS projects.

OSS is characterized as intensely people-oriented [271] and a knowledge-intensive software development process [465]. As open source software often relies on the volunteer efforts of software developers, the survival and well-being of OSS projects often depend on attracting contributions from the software community [98]. The OSS development model allows developers to integrate with non-technical members to form a broader, more transparent community [463]. It is also a model for the creation of self-learning [464] and self-organizing communities [449]. In this context, users and developers coexist in a community; working on the project based on personal needs and benefits, and by so doing, they acquire knowledge associated with their profession. The benefits include fun, reputation, learning, intellectual stimulation, improving skills, self-marketing and peer recognition [11, 258, 375, 466]. Membership in the community is fluid; current members can leave the community, and new members can join at any time [406]. Consequently, individual ownership of products is not apparent in OSS communities; instead, recognition of expertise is important. Community members believe in shared risks, shared rewards, and shared ownership [505]. This results in a strong culture and group behavior that have been developed in connection with the community [159].

2.6.3 OSS Security

A long debate has been going on in the security research community, whether OSS should be considered more or less secure than closed source software [73, 83, 368, 400, 413]. This debate has not led to any definitive conclusion so far [329]. Nevertheless, it is worth noting that the number of vulnerabilities found in OSS has increased by 371% since 2014, according to the State of Open Source Security Report [420]. As discussed earlier, one characteristic of OSS is the public availability of source code, including potential criminals and attackers. Attackers are able to study source code and exploit vulnerabilities that may be due to programming erros much more quickly. In addition, open source applications are usually developed jointly by volunteer contributions from groups and communities over the Internet. Attackers might also be able to contribute parts of the code to the software this way. Since OSS gives both attackers and defenders greater analytic power to do something about software vulnerabilities, the OSS communities need to adopt robust security practices that blend appropriate processes, methods and technologies.

[35] Concurrent Versions System (CVS) is a repository to store the software code produced by the developers

OSS development, being more distributed and less conventional, does not always go through a security process [84]. In OSS, the requirements usually are not well defined as proprietary software, which results in developers' lack of thorough understanding of system specifications and easily introduce more bugs in OSS [274]. Saleh M., et al [13] empirically examined a variety of OSS systems used for mobile computing and found that a majority of developers does not understand the real threats imposed by those vulnerable functions [13] while there was much literature devoted to addressing the problems of how to fix vulnerabilities in OSS development. These vulnerabilities might be introduced due to developers' non-awareness, bad programming practices or lack of knowledge against security vulnerabilities [10]. Although security review methods have been widely adopted in OSS projects including running code scanners [52, 97, 306], reviewers with the right security expertise in the problem domain are not easy to come by [504], and the people looking at the code may not be experts or understand the code fully, which could let more security bugs go unnoticed [83]. With today's increasing importance and complexity of OSS, the lack of knowledge and skills relevant for OSS developers to secure software development will result in more breaches that are serious in the future.

2.6.4 Learning in OSS Development

Learning in open source communities have been broadly studied in the literature. Hemetsberger and Reinhardt [197, 198] examined how knowledge sharing and learning processes develop at the interface of technology and communal structures of an OSS community. They suggested that knowledge is shared and learned in OSS communities through the establishment of processes and technologies that enable virtual re-experience for the learners at various levels. They viewed learning in OSS communities as experiential learning whereas learning is a process whereby learning is created through the transformation of experiences as developed by Kolb [247]. Au et al. [25] explored open-source debugging as a form of organizational learning, which heavily relies on adaptive learning [445] to overcome the complexity of software. Singh and Holt [416] provided insights on how the OSS community uses the forums for learning and solving problems. They explored the motivations for joining OSS communities, the learning that occurs in the communities, and the challenges to learning. Hardi [190] had a case study using the Google Chrome project to affirm that situated learning [263] is present among open source developers at an earlier time of a project. Sowe et al. have introduced a knowledge-sharing model to develop an understanding of the dynamics of collaboration and how knowledge is distributed over OSS development teams [422, 423]. Chen, Xiaohong analyzed key factors affecting knowledge sharing in OSS projects, which include participative motivation, social network, and organizational culture [76, 77].

Many OSS proponents believe that the OSS community offers significant learning opportunities from its best-practices [204, 257], which are different from the education of the traditional model [71, 144]. Although rapidly growing the current number of studies on learning in OSS communities, studies on the fields of software security are

scarce. This thesis filled this research gap by empirically explore the factors that affect learning about software security in OSS development, as well as the relationships among them. The corresponding research work is presented in RP III and RP IV.

Chapter 3

Research Design and Methodology

The main research methodology used in this thesis is the Design Science Research (DSR) methodology, in conjunction with the Design Theorizing Framework provided by Lee, Pries-Heje, and Baskerville [269]. This chapter is organized as follows. In section 3.1, an overview of DSR is presented, followed by theorizing in DSR in section 3.2. Section 3.3 presents the DSR process model proposed by Peffers et al. [349]. In section 3.4, the research design of this thesis is explained.

3.1 Design Science Research

The word *design* comes "from the Latin désigńare, which means to point the way" ([361]. p. 4). Design is "the use of scientific principles, technical information and imagination in the definition of a structure, machine or system to perform pre-specified functions with the maximum economy and efficiency" [472], which is "the core of all professional training; the principal mark that distinguishes the professions from the sciences" ([414], p. 67). DSR "addresses important unsolved problems in unique or innovative ways or solve problems in more effective or efficient ways ([203], p. 81); it seeks to create innovative options that are filtered and excluded until the design's requirements are fulfilled [202]. Iivari [214] defines DSR as "a research activity that invents or builds new, innovative artifacts for solving problems or achieving improvements, i.e. DSR creates new means for achieving some general goal, as its major research contributions. Such new and innovative artifacts create a new reality, rather than explaining existing reality or helping to make sense of it [existing reality]" (p. 4). The creation of innovative artifacts relies on existing kernel

theories that are applied, modified and extended through experience, creativity, intuition and problem solving [286, 472].

The study objective of design science is the creation and use of artifacts that can advance individual as well as organizational and societal flourishing. In design science, artifacts represent general solutions to a class of problems [34]. The different kinds of artifacts to be developed in design science have been stressed by March and Smith [283] who identify four different types of artifacts. According to them, design science research outputs comprise constructs, models, methods, and instantiations. Constructs represent a vocabulary of a domain and provide the means to describe problems that have been identified by the researcher. To bring structure to the problems identified within this domain, models provide a basis to describe and explore the relationship between different constructs of interest. To focus on those issues being most relevant to address the problem, abstraction and simplification are indispensable here. Using existent constructs such as algorithms define a sequence of steps to be taken in order to perform a specific task. Finally, an instantiation represents the realizations of an artifact that demonstrates the feasibility and applicability of designed models and methods. Hevner and Chatterjee [202] extend the understanding of the IT artifact and point out that, besides constructs, models, methods, and instantiations, DSR also aims at developing another type of artifact: better design theories.

The research in the field of software security is of an applied nature, which employs the theories from natural sciences, social science, and computer sciences to solve the problems at the intersection of information systems, information technology (IT) and organizations. To cater the nature to the applied and interdisciplinary nature of information systems research, DSR has gained acceptance in the research community. DSR combines applied design with the generation of theoretical knowledge in the pursuit of problem-solving. It tackles real problems that rarely have optimal solutions, and instead defines and pursues goals that provide satisfactory solutions [497]. DSR is an accepted and well-established methodology in the domain of information system research [284, 349, 448] and provides "a set of synthetic and analytical techniques and perspectives for performing researches in IS" [448]. Thus, based on the research objectives and considering the practical tasks when designing artifacts, DSR was adopted for the main research methodology as an overall research design in this thesis-

3.2 Theorizing in DSR

The most basic purpose of DSR is to create a novel and useful artifact and an accompanying descriptive design theory [203]. Walls et al. [472] and Markus et al. [286] illustrate that theorizing represents a fundamental activity in design-oriented research. Theorizing refers to the process of constructing a theory [479]. Theorizing may be a form of disciplined imagination in which concurrent trial-and-error thinking is iterated through imaginary experiments [479]. The main tenet of DSR is

that knowledge and understanding of a practical problem and its solution can be acquired through the creation and use of an artifact. Because DSR solves real problems that are implicitly motivated to solve a specific, it produces prescriptive rather than descriptive theory, methods or prescriptions that answer 'how can we reach the goal' ([472], p. 36). While theories of description tell us about the states of a system, theories of prescription tell us how to transition from one system state to another. Prescriptions have a goal state and constraints for when they are applicable. Creating prescriptive theories is a formalization of general problem-solving.

Fischer and Gregor [147] propose the Idealized Model for Theory Development (IM4TD), which suggests how scientific knowledge is created in DSR. The IM4TD makes a fundamental distinction between the context of discovery (identifying and capturing novelty) and the context of justification (validation as a scientific method). The IM4TD identifies three forms of theorizing strategies—deduction, induction, and abduction—that are used in both contexts. Deductive theorizing derives a conclusion by generalizing the existing theory to specific instances [269]. For instance, (a) premise: failure to incorporate user requirements leads to low user satisfaction, (b) instance: a system has failed to incorporate user requirements, (c) conclusion: the users of this system have low satisfaction. Falsification is the main mechanism of deductive theorizing. This means that a "theory can only be shown to be wrong, but never be proven to be right" ([269], p. 3). Theorists using a deductive approach deduce hypotheses from general knowledge and attempt to falsify them in a variety of settings; thus, a surviving theory is deemed to become more complete. As Fischer and Gregor [147] put it, deductive theorizing refers to the process whereby a specific conclusion can be logically deduced from one or more general theories or principles. They note that deductive theorizing is always "firm"—meaning that, if the theory is true, a logically deduced conclusion is necessarily true.

In contrast, inductive theorizing involves the formulation of a general proposition based on a particular proposition. For instance, (a) instance: every system that failed to incorporate user requirements has resulted in low user satisfaction, (b) conclusion: failure to incorporate user requirements leads to low user satisfaction. In other words, researchers make their observations based on sample instances of the population and generalize these observations to all entities of that population [147]. This form of theorizing develops general conclusions from particular cases; it builds theories from specific instances [269]. Schilpp [394] defines induction as "inference from repeatedly observed instances to as yet unobserved instances" (p. 211). Inductive theorizing is recognized as a valid theorizing method by modern researchers [127, 265, 276].

Abductive theorizing is commonly referred to as inference to the best explanation. It involves drawing a possible precondition from a specific consequence. For instance, one might conclude that (b) failure to incorporate user requirements leads to low user satisfaction from the specific instance that (a) a newly developed system did not lead to high user satisfaction. Abduction is a creative process and plays a vital role in introducing new ideas or hypotheses [147]. According to Charles S. Pierce and

Norman R. Hanson [189, 351], the abductive activity of creating a theory is based both on real-world observations that are inductively observed as well as theoretical viewpoints, premises, and conceptual patterns that are deductively inferred. Peirce [350] argues that "abduction is, after all, nothing but guessing" (p. 137) in that its goal is to derive a possible conclusion in terms of what can be possibly true as opposed to declarative logic whose goal is to determine a proposition to be true or false.

While these deductive and inductive approaches are a useful theorizing tool for theory development, theorizing for design often necessitates the adoption of a line of theorizing that is essential for problem-solving, i.e., abductive theorizing. DSR is a research methodology that generates both theory and real-world solutions to real-world problems and is particularly effective when abductive logic is required to reach acceptable solutions. Because it tackles real problems that rarely have optimal solutions, the adoption of abductive theorizing for design theorizing enables the search for a satisfying solution for a given design problem. This theorizing process provides a good example of disciplined imagination involving intuitive and creative thinking processes ([269] p.13). The theories produced are prescriptive, meaning that they describe methods for achieving goals. Because of this prescriptive style, DSR studies rely heavily on history and context to convey the situations to which their findings are applicable and to indicate how generalizable they are.

3.2.1 Design Theorizing Framework

Theorizing in this thesis is influenced by the research framework outlined by Lee et al. [269]. This theorizing framework provides a useful organizing mechanism to structure discussion and terminologies for distinguishing between activities that occur in the intervention occurring in abstraction and theorizing. In their framework, there are four entities: abstract problems, abstract solutions, instance problems and instance solutions (Figure 3.1). These four distinct quadrants fall into two theorizing domains: the abstract domain and the instance domain. The abstract domain is typically reserved for scientific discussion using concepts and theories, while the instance domain describes the specific implementations and evaluations. The four entities are connected by four theorizing activities, solution search, de-abstraction, registration, and abstraction.

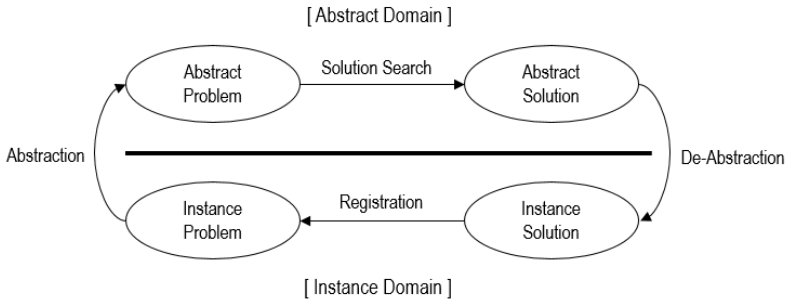


Figure 3.1: Design theorizing framework proposed by Lee et al. [269]

Solution search identifies solutions to a given problem. Solutions are in the form of prescriptions, things we can do that affect the desired change expressed by the problem. Solutions may be imaginary ideas, unimplemented designs, or material constructs; and they are built from a combination of searching existing solutions, prior experience, and creativity.

Registration checks whether a solution works, both for the original problem case and similar problem cases. In the case of finding similar problems, registration includes what might be called 'problem search'. As with solution search, registration may be done in thought experiments or through material constructs. A solution may be registered to a very similar problem to the one it solved originally, or it may be registered to broader or more abstract classes of problems. Whether it fails or succeeds in registering, the theoretical body of knowledge is improved by attempts to register solutions.

Abstraction attempts to make a theory more generalizable by taking a prescription and discarding detailed information, leaving only universal elements behind. The abstraction of design problems is a subjective process, in which the theorist uses their expertise to judge which of the elements are generalizable or essential elements of the problem and which are particulars of the instance problem. The abstraction of problems generally leads to kernel theories, those that govern a component of the design process at a high level. While not necessarily of practical use, these kernel theories provide the base for less abstract theories and methods.

De-abstraction instantiates abstractions for a particularized setting to ensure that they are valid in practice. This grounding may be done through creation, thought experiments, or comparison with other designs. This de-abstraction involves adding details pertaining to a specific context in which the solution will be applied, and all the details of the instance solution become articulated [269]. Each application in a different context demonstrates a method's effectiveness and supports the theory's generalizability.

There is no universal starting point in the design theorizing framework. All four of these activities may take place as human thought, it may be possible that these occur not cyclically, or in the order implied by the arrows, but perhaps may arise simultaneously ([269], p7). Intuitively, one might think that theorizing starts with the acknowledgment of an instance problem, and proceeds in the following order; identification of an abstract problem, development of an abstract solution, particularizing an instance of this solution and registering it to the originating instance problem (as represented in Figure 3.1). One can also start from the opposite side – design an abstract solution for the problem unknown and search for an instance where such an abstract solution can be applied to transform this instance into a better one.

While this thesis employs most of this framework unmodified, several alterations were made to better fit the nature of the theorizing process of the thesis. The revised

version of the design theorizing framework is depicted in Figure 3.2. First, while we agree that 'abstract' and 'instance' are two distinct domains, we discard this binary-categorization. Instead, the framework is amended as a four-quadrant scheme, in which each quadrant represents a specific domain. We argue that problems may not be specialized in the same solution domain during 'solution search' and 'registration' process, in the abstract or instance levels respectively. For example, researchers can search a different, in some respect, domain, where the solution may work for the problem. The second revision has based the recognition by Lee et al. that the arrows in their diagram are likely out of order, and not serial ([269], p7). In the revised framework, 'abstraction' and 'de-abstraction' are both reflected in the problems and solutions domains to increase the flexibility of theorizing activities.

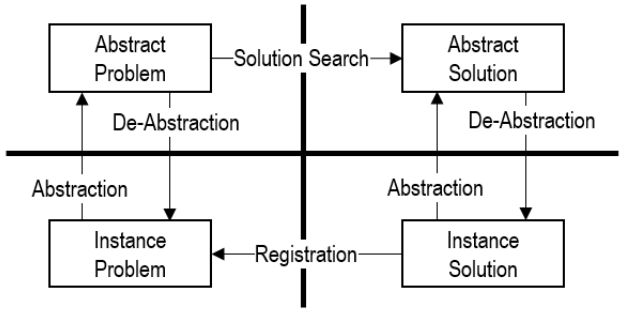


Figure 3.2: Design theorizing framework based on Lee et al. [269]

3.2.2 Theorizing in the thesis

In this research work, de-abstraction played a more significant role than an abstraction. Abstract problems and their accompanying solutions can show applicability across widely different settings. However, abstractions are difficult to develop and may not apply well to highly specific cases. With de-abstraction, an abstract problem is applied to identify an instance problem, and an abstract solution is applied to develop an instant solution. Figure 3.3 depicts the theorizing process in this thesis.

The thesis developed a design theory that proposes alternative learning approaches to fostering effective learning of security knowledge. The research work begins with the recognition of an abstract problem; that is, ineffective learning of security knowledge in software development. In the work associated with this theorizing framework, theorizing first moves to an instance domain when the problem is specialized in a real-world setting, the OSS development environment. This de-abstraction process goes on with a literature review, followed by contextual analysis in OSS communities, which both can be viewed as the systematic exploration of the instance problem space. By identifying limitations and opportunities in this specific situation, this work examines the socio-technical factors of security learning and makes suggestions for fostering a more effective security-learning environment. The

results in this theorizing step were organized in RP I, II, III and IV, and have shared with research communities.

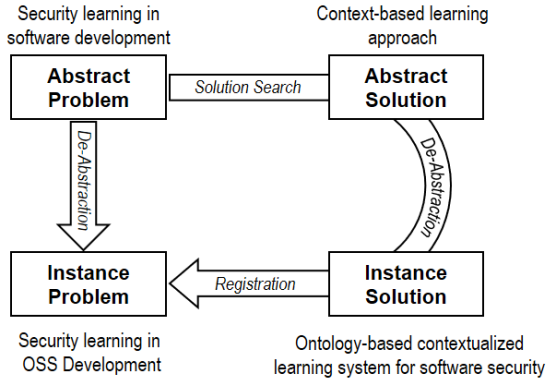


Figure 3.3: The theorizing process in the thesis (Adapted from Lee et al. [269])

After identifying problem domains, abstract and instance, the theorizing activity then crosses in the abstract domain as a search for universals in software-security learning. The result suggests a context-based learning approach for software security, with three guiding strategies that explain how security-learning improvement generally progressed in the pedagogical setting presented (published in RP V). This novel learning approach is then instantiated with an ontological knowledge base, as well as a contextualized learning system for software security. The results were published in RP VI and VII. After the realization of the designed artifacts, the theorizing process returned to the instance problem where it is registered against the learning system with a two-stage evaluation: a university learning environment, and OSS development settings. The former was published in RP VIII, while the latter published in RP IX. Table 3.1 presents an overview of the theorizing activities in the thesis and corresponding research papers.

3.3 DSR Process Model

According to Peffers et al. [349], for DSR, a methodology should incorporate three major elements: conceptual principles to define what is meant by DS research, practice rules, and a process for carrying out and presenting the research. (p. 49). The principles behind conducting design science research are to create and evaluate IT artifacts that may include models, constructs, methods and instantiations for solving research problems [203]. For practices, a methodology element requires the development of IT artifacts based on a research process that comes up with a solution by using existing theories or literature of a defined problem [153, 202]. Procedures are another important element of a methodology, which provides a generally accepted process for doing design science research [349]. Further, these IT artifacts are evaluated concerning their effectiveness and efficiency to improve performance in the development and use of information systems in many domains [284].

Table 3.1: Descriptions of theorizing activities in the thesis and corresponding research papers.

From domain	To domain	Theorizing activity	Description	Research Paper
Abstract problem	Instance problem	De-abstraction	Explored contextual factors of security learning in the OSS development environment.	I, II, III, IV
Abstract problem	Abstract solution	Solution search	Identified and proposed a 3-strategies context-based learning approach.	V
Abstract solution	Instance solution	De-abstraction	Developed into an ontology-based contextualized learning system.	VI, VII
Instance solution	Instance problem	Registration	Evaluated in both pedagogical and OSS development environments.	VIII, IX

For communication and to provide a comprehensible level of rigor in the design description, we followed the Design Science Research Methodology Process Model (DSRM process model, depicted in Figure 3.4) provided by Peffers et al. [349]. The DSRM process model is a useful synthesized general model that is derived from other models [176], which also provides a pragmatic and disciplined outline of the main considerations for successfully conducting DSR. The model accomplishes two things: it provides a road map for researchers who want to use design as a research mechanism for information science research; it may help researchers by legitimizing their research using understood and accepted processes [349]. This process model used a consensus-building approach, which ensures that this model is based on common process elements, discussed earlier in the literature related to design science research [349].

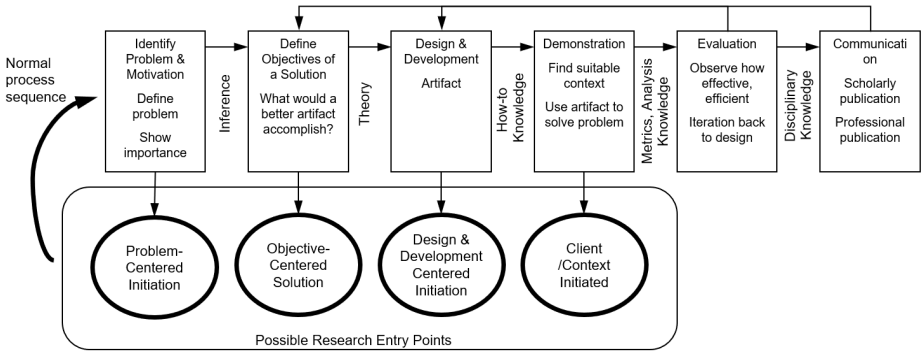


Figure 3.4: DSRM process model proposed by Peffers et al. [349]

DSRM process model distinguishes six activities and four different entry points to the design-science research process. The first entry point is the traditional problem-centered initiation, which is similar to qualitative and quantitative research methodologies. The second is the objective-centered solution approach, which enables researchers to approach the research endeavor by first setting objectives that can be quantitative or qualitative with the main idea of establishing how the new artifact is expected to support solutions to achieving the stated objectives. The third entry point is design-centered, where initiation can be a result of an interesting design or development problem. The fourth entry point is where the design starts with a research client.

The activities of the DSRM process model are: (1) identify problems and motivation, (2) define objectives of a solution, (3) design and development, (4) demonstration, (5) evaluation, and (6) communication. While considering these activities, there is no compulsion for researchers that they would always follow to a sequential order from activity 1 through activity 6. Instead, they may start at almost any step and move outward [349].

Activity 1 – Problem identification and motivation

In this activity, the specific research problem is identified, and the value of solutions is justified. In order to capture the complexity of problems and provide effective solutions, it suggests that the problem should be atomized conceptually [349]. Justifying the value of a solution provides two things. One, it motivates the researcher and the audience of the research to pursue the solution and to accept the results. Two, it helps to understand the reasoning associated with the researcher's understanding of the problem.

Activity 2 – Define the objectives for a solution.

Based on the evidence, reasoning, and inference, the process continues toward defining the objectives of a solution to solve the research problem. The objectives in this activity can be qualitative, in which description about the new artifact is expected to support a solution of a given problem, or quantitative, in which terms of how a desirable solution would be better than recently designed ones if there are any [349]. For this activity, knowledge of the current state and current solutions is required. The result leads to knowledge of the theory in the given field of research.

Activity 3 – Design and development

The design and development activity creates an artifact that addresses the explicated problem and fulfills the defined objective. A design research artifact can be any designed object in which a research contribution is embedded in the design [349]. This activity involves designing and developing an artifact that deals with the desired functionality as well as its architecture and then creating the actual artifact. Design and Development do not primarily aim to answer questions by producing

prescriptive or explanatory knowledge. Instead, its main purpose is to produce prescriptive knowledge by creating an artifact, the "How-to" knowledge about the design decision taken and their rationale [223, 349]. As Johannesson and Persons comment on research strategies for Design and Development: "…it is not critical that research methods are used for devising possible solutions, but that any approach for generating solutions is admissible, as long as it works." ([223], p. 125)

Activity 4 – Demonstration

The Demonstration activity involves the use of the artifact to solve one or more instances of the problem by experimentation, case study, proof, simulation, or other appropriate activity [349], thereby providing the feasibility of the artifact. A demonstration shows that the artifact can solve some aspects of a problem in one illustrative or real-lift case, which can be seen as a weak form of evaluation as well [223]. A demonstration can also help communicate the idea behind the artifact to an audience vividly and convincingly. The output of this activity is a demonstrated artifact including information on analytic metrics of the artifact in one case. The generated knowledge is both descriptive and explanatory; the former describes how the artifact works in one situation, while the latter explains why the artifact works.

Activity 5 – Evaluation

The fifth activity of the process model is Evaluation, which determines how well the artifact is able to solve the given research problem and to what extent it fulfills the objectives. The evaluation activity aimed at rigorously providing essential feedback to the building and development processes by demonstrating utility, quality, and efficacy of the proposed framework [203, 515]. This activity compares the actual observed results from the use of the artifact in the demonstration with the solution objectives from activity 2. The result of the evaluation activity leads to disciplinary knowledge, which is an evaluated artifact including the information on the usefulness of the artifact. At the end of this activity, the researchers can decide whether to iterate back to the design and development activity for the effectiveness of the artifact or to continue to the last activity of this model [349].

Activity 6 – Communication

The objective of the Communication activity is to communicate the research problem and its significance to researchers and other target audiences such as practicing professionals [349]. In addition, the utility, novelty, and efficacy of a designed artifact are also shared among research communities. It not only enables practitioners to take advantage of the benefits offered by the given solution to a problem but also enables researchers to build a cumulative knowledge base for further extension and evaluation.

3.4 Research design in the thesis

This thesis followed Peffers et al.'s DSRM [349] to approach the development of the contextualized learning application for software security as a series of five iterations, with each iteration indicating a specific design cycle (see Figure 3.5). Hevner et al. ([203], p. 89) term this iteration the 'generate/test' cycle. The evaluation of our artifacts, as for most DSR that deals with human–artifact interaction, took the form of an experiment. In a DSR project, the research process frequently iterates between development and evaluation phases rather than flowing in waterfall fashion from one phase into the next [255].

After the identification of the research problem and motivation, given in sections 1.2 and 1.3 respectively, a five-iteration design activity was carried out, in which each design cycle (DC) contained the following steps: objectives for a solution, design and development, demonstration, and evaluation. Evaluations were done for each cycle, rather than only once at the end of the design process. Each design-cycle not only derives designed artifacts but also results in knowledge contribution through communication, which involves professional and scholarly publications and presentations [349].

DC 1: A Socio-technical framework for security learning in the context of OSS development

Drawing on Figure 3.5, the first design cycle concerns establishing a socio-technical framework of security learning in the context of OSS development. This design activity started with analyzing the existing body of knowledge on OSS security practices using the method of Systematic Literature Review (SLR). SLR study is a defined and methodical way to summarize the empirical evidence concerning treatment or technology, to identify missing areas in current research or to provide background in order to justify new research. It provides a much stronger basis for making claims about the research questions [230, 326]. Based on the identified and relevant articles, the result of the SLR study gave an insight into the gaps in the literature on socio-technical perspectives and knowledge management practices of OSS security. This step of the DSR process was addressed with research question 1.1 and outlined in RP I.

After SLR, two empirical studies were conducted to investigate the real-world problems and to identify prospectus, limitation, and uncertainty embedded in the security practices and learning of security knowledge in OSS development environments. The empirical study is a way to gain knowledge by the collection and analysis of primary data based on direct observation and/or measurement methods in the 'problem domain' [518]. The former refers to qualitative, and the latter refers to quantitative research methods [29]. Qualitative research methods are used to explore why or how a phenomenon occurs, to develop a theory, or describe the nature of an individual's experience, while quantitative methods address questions about

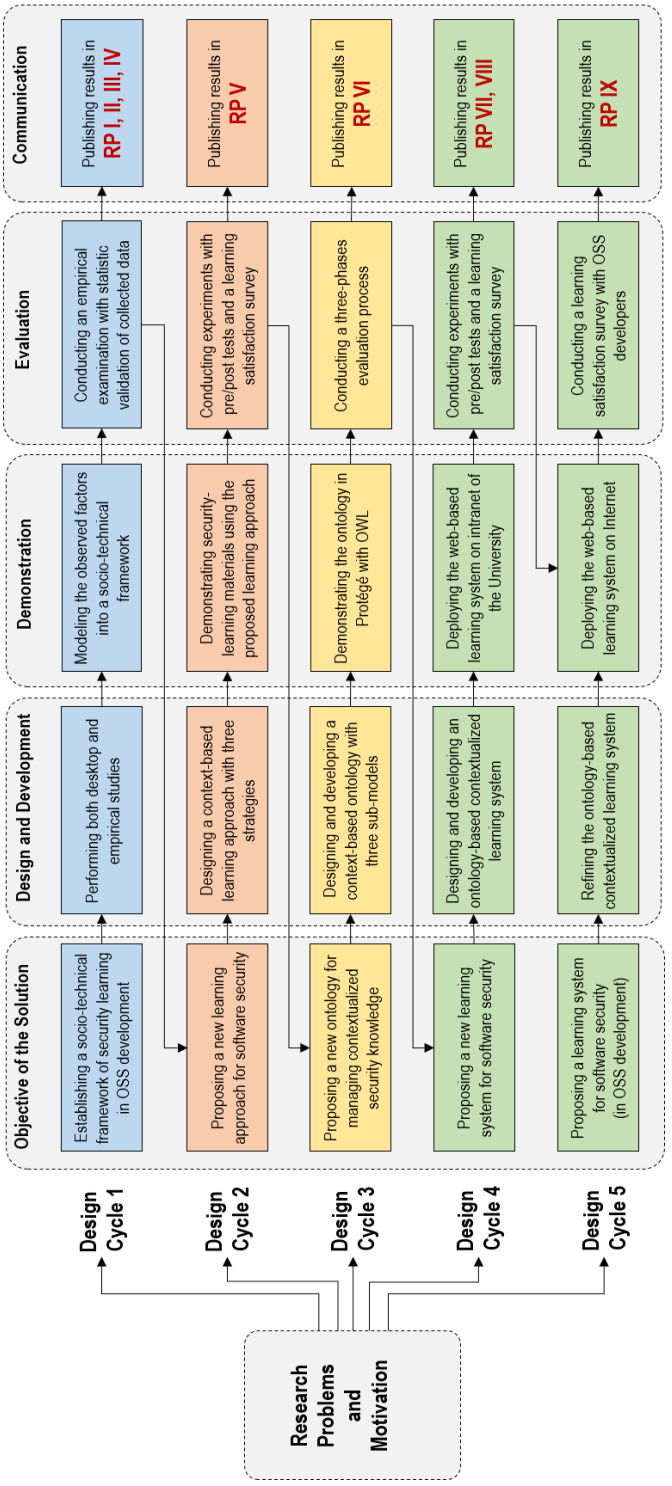


Figure 3.5: Iterations of DSR design cycles.

causality, generalizability, or magnitude of effect [146]. To answer RQ 1.2, a quantitative research approach was adopted, in which a questionnaire was prepared to gather information from OSS participants on the social and cultural aspects related to secure software development in OSS communities. The findings from the study were summarized and reported in RP II.

Further, to answer RQ 1.3, in the second study, a Mixed-Method Research (MMR) design was selected in order to broadly explore and understand the socio-technical aspects of security learning in the context of OSS development, as well as the interaction effect among the observed factors. MMR frequently referred to as the 'third methodological orientation' [436], draws on the strengths of both qualitative and quantitative research. While there is no universal definition of mixed methods research, Creswell and Plano Clark [95] outline its core characteristics: in a single research study, both qualitative and quantitative strands of data are collected and analyzed separately, and integrated – either concurrently or sequentially – to address the research question. Onwuegbuzie and Combs [337] concur, writing, "mixed analyses involve the use of at least one qualitative analysis and at least one quantitative analysis – meaning that both analysis types are needed to conduct a mixed analysis" (p. 414). Instead of approaching a research question using the binary lens of quantitative or qualitative research, the mixed methods research approach has the ability to advance the scholarly conversation by drawing on the strengths of both methodologies.

In MMR, qualitative data is first collected and analyzed, and themes are used to drive the development of a quantitative instrument to further explore the research problem [95, 337, 436]. As a result of this design, two stages of analyses were conducted: an exploratory stage and a confirmatory stage. The reason for employing an exploratory study in the first stage was that important constructs relate to socio-technical aspects of OSS development and their influence on security learning were unknown, and relevant quantitative instruments were not available. In the first stage, data were collected adopting a qualitative-ethnographic research method in the three selected OSS projects. Ethnography focuses all the details of what members of culture do in their daily actions since culture is enacted through these details [18, 407]. Specifically, this study employs a socio-technical systems approach to systematically and holistically take into account the social context as well as technological aspects of the studying subjects. In fact, a socio-technical perspective can provide a stronger framework than any other approach because of its integrative and holistic nature [279]. With the identification of socio-technical challenges, the study then examined the main factors that were once disproportionately considered in the learning of security knowledge in OSS development. This study resulted in a conceptual socio-technical framework, which describes the interrelationship among social aspects (cultural and structural), and security knowledge sharing and learning behavior. This conceptual framework accompanying seven hypotheses was validated through an empirical examination, including the questionnaire design, data collection, and statistic correlation and linear regression analysis from 324 valid questionnaires. The

findings from the exploratory stage were reported in research paper III, while the results of the confirmatory stage were reported in RP IV.

DC 2: A context-based learning approach for software security

The primary objective of the second design cycle was to propose a novel method of artifacts for structuring and presenting software security knowledge, answering research question 2.1. To this end, a context-based learning approach was first proposed, adopted from concepts of CBL and literature from psychology and education. The artifact was further evaluated to prove the effectiveness in improving learners' learning outcomes on studying software security, answering research question 2,2. A two-round experiment was conducted with 42 Bachelor students to evaluate the effectiveness of the proposed learning approach versus conventional learning materials. The method of experiments allows researchers to achieve high internal validity by carefully controlling the conditions under which an experiment is carried out [223]. Two types of the instrument were designed and built in the data collection scheme, including (1) pre-tests and post-tests for measuring knowledge gain, and (2) survey questionnaires for measuring learning satisfaction. After the design and evaluating this iteration, this work was communicated to the research community with RP V.

DC 3: A context-based ontology for managing contextualizing security knowledge

Taking the proposed learning approach into further design consideration, DC 3 focused on the artifact of the ontological knowledge model, addressing RQ 3. The ontology is a key component to model the security knowledge and to support the development of the learning system, so having a distinct design cycle to validating this component was necessary. The objectives of DC 3 were three-fold, (1) to design and construct an ontological knowledge base to manage contextualized knowledge, (2) to validate the feasibility of ontology, and (3) to visualize the knowledge representation as a pre-study for DC 4. In accordance with the strategies in the proposed learning approach, the design of the ontology was composed of three modeling activities: application context modeling, domain knowledge modeling, and contextualized knowledge modeling. The ontology was constructed and demonstrated in Protégé editor and validated through a three-phase evaluation process: domain expert evaluation, competency question evaluation, and application-based evaluation. The design, development, and evaluation of the ontology in this design cycle were summarized in RP VI.

DC 4: An ontology-based contextualized learning system for software security

The objective of the fourth design cycle was to develop a contextualized learning system for software security. This artifact was designed as a proof-of-concept to security educators regarding an ontology-driven web application for context-based learning, integrating the designed artifacts from DC 2 and DC 3. The former suggests

the representation of security knowledge and the embedded learning process; while the latter acts as the kernel knowledge base. While the system architecture was inherited from the DC 3, a major alteration was made in the knowledge layout to facilitate context-based learning appropriately. To achieve this, the proposed learning strategies were adopted in the user interface for structuring security knowledge. The design and development activities of the artifact in this design cycle were summarized in RP VII, which answered research question 4.1.

Furthermore, to address research question 4.2, the learning system was deployed in the intranet of the university environment and evaluated through a controlled experiment with 36 students. This evaluation activity compared the learning outcome between two groups of students, the control group and the experimental group; the former used the conventional learning materials, while the latter used the proposed learning system to study the assigned topics. To measure dependent variables, two types of instruments were used: (1) knowledge test sheets, for measuring knowledge-gain, and (2) a survey questionnaire, for measuring learning satisfaction. The proof-of-concept of the innovative artifact, including the experimental evaluation methodology were summarized and reported in RP VIII.

DC 5: Validation in OSS development environments

A generic solution for real-life problems cannot be proven formally, but requires testing via the implementation of the solution in one or more situations and investigation whether it solves the intended problem or not [41]. To further answer research question 4.3, in this design cycle, the proposed artifact was first refined to improve the usability, including the appearance of the concept map and the dynamic layout. Afterward, it was validated by OSS developers through the actual deployment of the learning system on the internet. This would enable the artifact to go from a proof-of-concept to a more generalized proof-of-use and proof-of-value assessment [333]. The objectives of the fifth design cycle were two-fold: (1) to test and validate the beta version of the learning system with software developers in OSS development projects, and (2) to conclude the effectiveness of the proposed security learning system and lessons-learned. For the demonstration, the ontology was prepared with actual software scenarios that were manipulated from a homegrown web application by the author, an e-Store application with PHP and Java programming languages.

In this evaluation study, an online questionnaire was created to collect individual-level perception data about his/her experience in using the learning system with both quantitative and qualitative questions. The quantitative questions dealt with the aspects of system features and the embedded learning approach, while the qualitative questions asked participants to share their thought about the weakness and strength on all aspects of the system. Through sending research invitation letters, a total of 21 developers on GitHub were recruited for the artifact assessment. After the evaluation in the design cycle, this work was communicated to the research community by

publishing it in the RP IX. Table 3.2 is a mapping table, whereby the applied research methods, DSR activities, and the published research papers are mapped against the corresponding research questions.

Table 3.2: Mapping table for research questions, applied methods, DSR activities, and research papers

Research Question	Research Method							Theorizing Activity	DSR Activity	Research Paper
	QaR	QnR	MMR	SLR	Survey	Experiment	Case Study			
RQ 1.1				✓				De- abstraction	DC 1	I
RQ 1.2		✓			✓			De- abstraction		II
RQ 1.3	✓	✓	✓					De- abstraction		III, IV
RQ 2.1	✓							Solution search	DC 2	V
RQ 2.2		✓			✓	✓		Solution search		
RQ 3	✓						✓	De-abstraction	DC 3	VI
RQ 4.1	✓							De-abstraction	DC 4	VII
RQ 4.2		✓			✓	✓		Registration		VIII
RQ 4.3	✓	✓			✓			Registration		IX

Chapter 4

Summary of Included Publications

This chapter presents extended summaries of the included research papers published in the peer-reviewed professional and academic international conferences and journals in software security and security education. Each paper is presented followed an IMR format: **I**ntroduction, **M**ethodology, and **R**esult. Full versions of the research papers are given in Part II of this thesis.

4.1 (RP I) Software Security in Open Source Development: A Systematic Literature Review

4.1.1 Introduction

Many security studies have been conducted by both researchers and practitioners on the mechanisms of building security in OSS development. However, the number of new vulnerabilities keeps increasing in today's OSS systems. The essence of this research was to identify areas for possible improvement or enhancement via systematic evaluation of relevant and current security studies in the context of OSS development as reported in the literature.

4.1.2 Methodology

In this paper, a Systematic Literature Review (SLR) was carried out to extract security studies conducted in the context of OSS development from the year 2000 to 2016. Through a four-stage selection execution process (depicted in Figure 4.1), a total of 42 papers were selected. The selected papers were classified and analyzed using the

OWASP Software Assurance Maturity Model (SAMM) and Socio-Technical Security Framework.

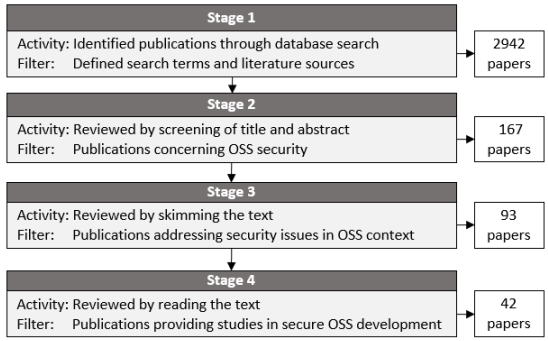


Figure 4.1: Paper selection process of SLR

4.1.3 Result

Based on the results of the SLR, the following findings were concluded. First, the areas of Construction and Verification (Secure Architecture, Code Review, and Security Testing) are the most cited in security studies, while Governance and Deployment received the least attention in the selected studies. Second, the discussion of technical aspects has happened in 98% of the selected studies (41 out of 42), However, less than 50% of studies talked about the social sectors of OSS security. There is an obvious dearth of research on the social-technical perspectives of OSS security. Third, there are no OSS security studies addressing security issues from the educational and knowledge management aspects. A closer look at the aspects of security knowledge management and learning seems to be needed in OSS development. As the diversity of OSS products and projects increases, there will no longer be a single technical approach for achieving optimal software security in all OSS projects. This study suggested that future researchers should explore approaches from socio-technical aspects in helping OSS developers learn the necessary security knowledge to fulfill the need of their work, further, to reinforce their behaviors towards OSS security.

4.2 (RP II) An Empirical Study of Security Culture in Open Source Software Communities

4.2.1 Introduction

OSS communities, with their complex network of interactions between people and people, people and processes as well as between people and things, represent unique characteristics, technical and non-technical. This socio-technical perspective suggests a deeper analysis of the relationship between culture, methods, tools, development environment and organizational structure. The result of this type of analysis can be

used to improve process performance, disseminate best practice and generate artifacts. As there is still a dearth of empirical research on the social study of OSS security, mentioned in RP I, this study intended to complement the research by empirically investigating the social and cultural aspects of OSS security. By exploring the current security culture in OSS communities, this paper provides an in-depth understanding of the influence of security on participants' security behaviors and organizational decision-making.

4.2.2 Methodology

This paper adopted a quantitative approach with a survey instrument to investigate OSS security culture. It first established the research framework for security culture with identified six dimensions: attitude, behavior, competency, subjective norms, governance, and communication. The framework was then used as the theoretical foundation to design a security culture questionnaire. Overall, 254 respondent questionnaires were used for the statistical analysis.

4.2.3 Result

Figure 4.2 depicts an overview of the security culture scores. Attitude is the only dimension that reaches a mean value at a degree of 4.00. The respondents overwhelmingly reported a positive attitude toward software security. More concerning, however, is the evidence that a significant minority of respondents were unwilling or unable to put this positive attitude into practice. The behavior of OSS participants is at a mild level of maturity, but still, on average, insecure. This study also revealed a missed set of means in terms of security practice reinforcement and demonstrates a clear knowledge gap that must be addressed by OSS communities. Notably, this study revealed there were weak subjective norms and security governance to support security culture, suggesting that limited development of trust and supportiveness between peers, as well as an insufficient complement to security expertise. Last, communication of security information is the least developed dimension in security culture, as the mean is the lowest of all six dimensions.

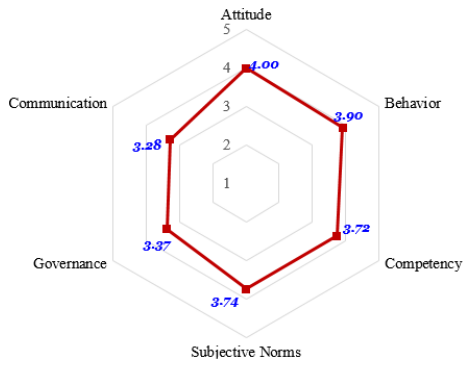


Figure 4.2: The mean score of security culture dimensions

This study indicates that OSS communities still have some way to go in ensuring that software security is high on the list of project priorities, gets participants' attention, and strengthens participants' competency in promoting positive security best practice. With respondents' broadly positive attitude to security, OSS communities clearly need to place more focus on providing members with information related to security subjects, offering opportunities for learning and supporting self-development of security knowledge. One is to provide dedicated communication channels to ensure that participants have reached security information, the codification knowledge when they need it, and importantly, are aware of where they can locate it. With just a glance, participants understand they need to pay attention and take any recommended action immediately. Through this structural mechanism, the security knowledge gains valuable insights from the community, and further, facilitating discussion and decision making and sharpening personalization knowledge.

4.3 (RP III) Learning Secure Programming in Open Source Software Communities: A Socio-Technical View

4.3.1 Introduction

Learning in OSS communities has been a major interest for researchers. Many OSS studies have indicated that the OSS community offers significant learning opportunities from its openness, transparency and collaboration phenomenon. However, existing research on security learning in OSS development has paid only modest attention to the task of integrating existing theory from relevant fields. Besides, there is a lack of depth in current understanding regarding security learning in OSS development. Therefore, there is a need for more integrative research in this field. This paper is the part one of the two empirical studies on security knowledge sharing and learning behavior in OSS communities, where the first paper explores the socio-technical factors of the problem domain, and the second investigates how these factors complement each other in shaping security knowledge sharing and learning behaviors in OSS communities. This paper presents an initial insight into present knowledge acquisition and learning about software security in OSS communities.

4.3.2 Methodology

This study utilized a qualitative/ethnographic approach to get an in-depth understanding of the socio-technological realities of the research subjects: the three OSS projects. The research findings were synthesized based on the socio-technical system model (Culture, Structure, Method, and Machine) to examine the socio-technical factors that are once disproportionately considered in learning about security in OSS communities.

4.3.3 Result

First, in the Method aspect, this study observed that security learning in OSS communities is two-fold: Self-directed learning and learning from the mistake. OSS developers learn software security by means of available security information on project websites. The code review process enablers for developers to reflect their code, take corrective actions and build concrete experiences. In the Culture aspect, the security culture backgrounds either at organizational or at the individual level have impacts on the amount of security knowledge transferred within the community, further, affecting participants' learning processes. If an OSS project truly holds a value that software security is important, then particular behaviors and actions can be expected among the participants. From the analysis of the Structure aspect, it observed that effective learning of software security results in coordinating necessary security expertise in the project, which enables a high level of security knowledge creation and the satisfaction of the learning process. A major problem found in this study is a lack of sufficient as well as efficient knowledge sharing and learning mechanisms for software security in OSS communities. The security knowledge is scattered over the community websites (source code, documentation, wiki, forum, conference pages, etc.), and the quantity of transferred knowledge is varied by projects. Finding and learning knowledge about secure programming becomes a key challenge that is highly dependent on the resources the community provides.

4.4 (RP IV) An Empirical Study on Security Knowledge Sharing and Learning in Open Source Software Communities

4.4.1 Introduction

This paper is part two of the studies on security knowledge sharing and learning in OSS communities. After the prior ethnographic study, we were interested in obtaining a deeper understanding of how the observed socio-technical factors complement each other in shaping security knowledge sharing and learning behavior.

4.4.2 Methodology

Based on the literature review, and our understanding of causal links between social and technical factors, we formed the hypotheses with a conceptual framework for security knowledge sharing and learning in OSS communities. Figure 4.3 depicts the conceptual and theoretical structure that includes four constructs, namely: security culture, expertise coordination, security knowledge sharing, and software security learning. We validated the hypotheses through conducting empirical examinations including a questionnaire survey, survey data collection, index measurement, validity, and reliability testing and linear correlation analysis among 324 valid questionnaires.

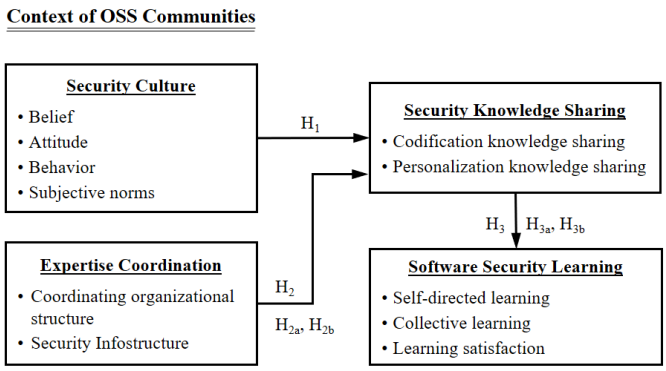


Figure 4.3: The conceptual framework for security knowledge sharing and learning in OSS communities

4.4.3 Result

With statistical analysis, all of the hypotheses were tested and approved (Table 4.1). Based on the result, social factors (culture and structure aspects) have significant impacts on the software-security learning behavior through the mediating variables of knowledge sharing when considering security culture and expertise coordination. Based on the results of the paper, we argued: (a) OSS communities should cultivate a security culture to promote the value of software security to their products. With such a phenomenon, OSS developers and users will be willing to share and talk about software security, and there will provide more opportunities to draw lessons learned from each other's experiences that should be actively taken into account in projects. (b) Having central security expertise responsible for security knowledge transfer, such as, specific security web pages can be included in the project website or repository, which will lead to security learning practices being established.

Table 4.1: Testing results of research hypotheses

Hypothesis.	Result
H_1. Security culture is positively associated with security knowledge sharing.	Supported
H_2. Expertise coordination is positively associated with security knowledge sharing.	Supported
H_{2a}: Coordinating organizational structure has a positive effect on security knowledge sharing.	Partially supported
H_{2b}: Infostructure has a positive effect on security knowledge sharing.	Partially supported
H_3. Security knowledge sharing is positively associated with software security learning.	Supported
H_{3a}: Codification knowledge sharing has a positive effect on software security learning.	Supported
H_{3b}: Personalization knowledge sharing has a positive effect on software security learning.	Supported

4.5 (RP V) Towards a Context-Based Approach for Software Security Learning

4.5.1 Introduction

Software security knowledge is multifaceted and can be applied in diverse ways. Learning software security is a complex and difficult task because learners must not only deal with a vast amount of knowledge about a variety of concepts and methods but also have to demonstrate the applicability of the knowledge through experience in order to understand their practical use. In traditional software security teaching, little attention is given to what the security knowledge really means to learners, and there is not much content addressing the connections between real-world situations and security concepts. To facilitate effective learning about software security, this paper proposed a context-based approach to structuring and presenting software security knowledge.

4.5.2 Methodology

The proposed context-based approach includes three main strategies. In this approach, teaching starts with an application context that has an orienting purpose. The design of the application context aims to activate the learner's prior knowledge of software programming and anchors the learning about security knowledge. The second strategy is to organize underlying security knowledge in a structured manner that can stimulate learners' mental models to support more efficient learning in the specified context. The third is to guide learners to engage with concrete knowledge before studying abstract knowledge. This strategy assists learners in discovering meaningful concepts and relationships between practical functions and abstract knowledge when working in this context. Furthermore, it helps them apply knowledge in various other contexts.

The approach was evaluated through a controlled quasi-experiment with 42 Bachelor students in the setting of a university learning environment. Two types of learning materials were designed in a printed format as the experimental treatments: one used a conventional approach, while the other type adopted the proposed context-based approach to organizing software security knowledge.

4.5.3 Result

From the results of the experiment, there were positive findings to the adoption of the context-based learning approach, in terms of two measurements: security knowledge gain and learning satisfaction. According to the result, the proposed approach can be regarded as a solution to problems faced in security teaching practices at school. This study showed that it is effective in terms of promoting students' achievement and developing better attitudes towards software security. It was also concluded that

students receiving context-based instruction retained what they learned more in the practical type questions. Thus, the context-based approach can be applied during software security or computer security courses to make students more competent and interested in security knowledge.

4.6 (RP VI) Managing Software Security Knowledge in Context-An Ontology-Based Approach

4.6.1 Introduction

Security has become an important part of today's software development projects. However, due to the diversity of software development projects, software developers not only require knowledge about the general security concepts but also need the expertise to deal with variant technologies, frameworks, and libraries that are involved in the software development process. Although much security information is widely available in books, open literature or on the Internet, the content is traditionally encapsulated in unstructured or semi-structured formats, and commonly organized in a security-centric way. It is difficult for software engineers to extract relevant pieces of knowledge and apply it to their application-specific decision-making situations.

4.6.2 Methodology

This paper designed and implemented an ontological knowledge base for enabling managing software security knowledge in the context of software applications. This ontology organizes security knowledge around contextual software scenarios linking to security knowledge, both practical and theoretical. The design of the ontology consists of three parts: application context modeling, security domain modeling, and security contextualization modeling. Figure 4.4 depicts the full view of the ontology-based knowledge model, including the interrelationships of the components. The application context model defines a complete representation of what context is in a particular domain. The security domain model describes the theoretical knowledge, which is of teaching subjections through a set of concepts: Security Attack, Security Weakness, and Security Practice. The security contextualization model manages security knowledge in the context of specific scenarios and brings together the conceptual knowledge that is described in the security domain model.

This ontology differentiates from other ontology work in the following aspects:

(1) The ontology is context-based, which models security knowledge with a diversity of software features and technologies;

(2) The ontology describes security knowledge with a contextual situation, and meanwhile, complements the concrete knowledge with abstract description.

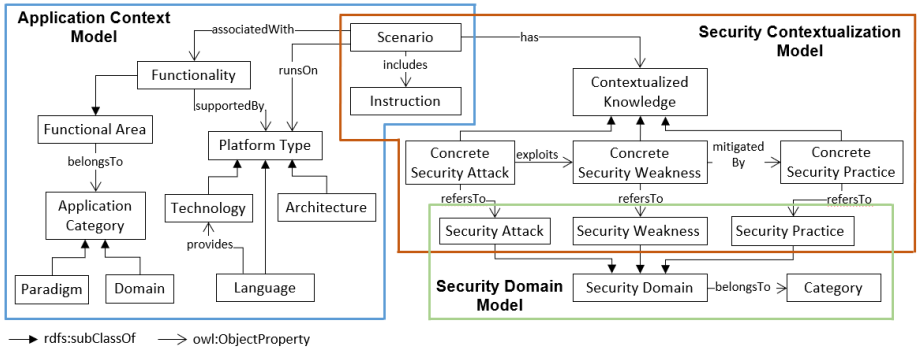


Figure 4.4: The ontology-based security knowledge model.

4.6.3 Result

This ontology was validated through a three-phase evaluation process. First, the ontology structure, including concept definitions and relations, were reviewed and analyzed by a security professional. Second, the ontology was evaluated with competency questions against its initial requirements. Third, the ontology was evaluated by being plugged into a web application to demonstrate the knowledge presentation. The evaluation result showed that the proposed ontology is deemed feasible in formalizing and managing contextualized knowledge in the security domain. In the practical software development process, software engineers are allowed to find solutions to exceptional situations by searching for similar contexts. On the other hand, in the pedagogical environment, a course tutor, who is engaged in the introduction of security vulnerabilities, can use the proposed ontology to quickly identify a number of real-world examples of facing a specific security attack or vulnerability, to improve the effectiveness of learning.

4.7 (RP VII) Development of Ontology-Based Software Security Learning System with Contextualized Learning Approaches

4.7.1 Introduction

Building secure applications is a complex and demanding task that developers often face, especially because the domain is rather context-specific, and the real project situation is necessary to apply the security concepts within the specific system. While learning software security, developers interpret security knowledge they gain with a range of strongly held personal programming experience. However, the traditional learning materials give little attention to what a real-world situation really means to developers, and there is not much content addressing the connection between security concepts and learners' prior knowledge. Consequently, the way that

developers process security information and their motivation for learning is not touched by conventional methods. Since software engineers are not experts on security in general, there is an ever-increasing need to help them learn security knowledge in a fashion manner. This paper proposes a learning system in the domain of software security, which aims to create conditions for more effective learning of software security that can motivate learners and stimulate their interests.

4.7.2 Methodology

The design of the security learning system was inherited from the previous research works, presented in Paper V and Paper VI: the former positions as a kernel knowledge base for the learning system, while the latter guided the user interface design of the knowledge presentation. This learning system facilitates the contextual learning process by providing contextualized access to security knowledge through real software application scenarios.

4.7.3 Result

A proof-of-concept prototype was developed based on the proposed learning approach and the ontological knowledge base. The front-end was designed as a web-based user interface with PHP and JavaScript libraries whereas the backend was implemented in Java, and using Jena API for accessing the ontology. The learning process begins with a selected contextualized scenario in the application context familiar to learners and then gradually leads to an understanding of the abstract part of security knowledge. To guide learners navigating through the contextualized knowledge efficiently, the knowledge content outlined in a graphical Concept Map. The corresponding security knowledge, contextualized and theoretical, was displayed interactively while learners click the node of Concept Map. In such an environment, learners discover meaningful relationships between the abstract explanation and the practical demonstration in the context of real software applications they are already familiar with; security concepts are internalized through the process of discovering, reinforcing, and relating.

4.8 (RP VIII) Preliminary Evaluation of an Ontology-Based Contextualized Learning System for Software Security

4.8.1 Introduction

This paper presents an initial investigation into the impact of the context-based learning approach in software security using the proposed learning systems.

4.8.2 Methodology

An experiment in a university learning environment was conducted to evaluate the effectiveness of the proposed learning system. This experiment employed pre-test/post-test and questionnaires to measure students' security knowledge gain and their learning satisfaction respectively.

4.8.3 Result

The result of the experimental evaluation showed the usefulness and feasibility because it shed some light on the potential benefits of context-based learning. First, with the support of contextualized learning, the experimental group students yielded significantly better performance than those in the control group in terms of security knowledge gain. Second, according to the results of the questionnaire survey, the students expressed higher learning satisfaction with the learning system using contextualized security knowledge than conventional learning materials. In addition, most students were very interested in the proposed learning system and all agreed that this approach could ease the information load effectively.

4.9 (RP IX) Learning Software Security in Context: An Evaluation in Open Source Software Development Environment

4.9.1 Introduction

As part of an investigation into context-based learning in the domain of software security, this study discovered and examined the impact of the developed contextualized learning system in software development environments. This paper presents an evaluation study with software developers in OSS projects.

4.9.2 Methodology

To evaluate the efficacy of the proposed security learning system, a questionnaire-based survey was conducted to collect OSS developers' perception of the system features and the embedded learning approach. A total of 21 voluntary participants from GitHub were recruited to participate in the system evaluation and completed the survey questionnaire.

4.9.3 Result

The results of this evaluation study indicated that the proposed learning system has the potential to be an effective learning tool that can motivate OSS developers to learn about software security. First, the respondents overall evaluated the practicality of

system features with a positive degree. They highly recommended the use of software scenarios with graphical and contextualized security knowledge presentation. Second, the results also revealed the learning approach kept developers interested and engaged. Consequently, they overwhelmingly expressed their satisfaction with the learning sessions. Based on the findings presented in the paper, the context-based approach is deemed as a suitable approach to support developers' security training and education in software projects.

Chapter 5

Summary of Contribution

The merits and benefits of conducting scientific research can be numerous for the researcher, the community, practice, and ever-developing science. This thesis contributes to the field of software development and security education. In particular, it sheds light on the OSS learning context and context-based software security learning. In this thesis, eleven research papers contribute to the domain knowledge base, and nine of these papers are included in the thesis. Table 5.1 summarizes the contributions of the thesis and the corresponding publications. Although the contributions were presented individually, the impact of this thesis is also due to a synergistic effect. To facilitate such an effect, Figure 5.1 presents the contributions in an integrated manner that links the research questions, DSR activities, corresponding artifacts, research papers, and Fenstermacher and Richardson's learning ingredients [142] (presented in section 1.3). The right-hand side of the figure addresses the primary and secondary ingredients of each contribution, highlighting how the joint set of papers addresses the problem, artifact design and development process, and artifact evaluation. For each relationship, an arrow points from the primary focus of the paper toward the secondary focus. In general, the primary focus is the study's aims and methods, while the secondary focus relates to the results of relevant studies or suggestions for expanding security learning.

Overall, the included research papers address the full range of ingredients in the framework. This chapter concisely describes the contributions in terms of their initial goals and the research questions.

RQ 1: How do socio-technical aspects affect individuals' learning of software security in the context of open source software development?

This question is addressed in the problem identification of DSR activities, a systematic literature review of security research in the context of OSS development, and

empirical studies on security learning in selected OSS communities. The three respective contributions (C1, C2, and C3) primarily focus on the ingredient of "social surroundings supportive of teaching and learning." Contribution 1 outlines the investigation and identification of the strengths and weaknesses of security practices for secure OSS development from the literature, encompassing both social and technical aspects. This contribution, published in RP I, points toward the ingredient of "opportunities to teach and learn" by providing software security researchers with a firm basis for developing new security approaches, addressing research gaps from both the socio-technical and knowledge management perspectives to open opportunities for learners to engage. Contribution 2, addressed in RP II, offers an empirical study on the social and cultural security aspects of OSS development. This contribution recommends cultivating and maintaining an OSS development community culture that values developers' positive security attitudes and behaviors. For this reason, C2 points toward the ingredient of "willingness and effort on the part of the learner," as it influences learners' motivation to obtain security knowledge. Contribution 3 provides an investigation of security knowledge acquisition and learning in OSS communities. This contribution points toward the ingredient of "high-quality teaching" since the knowledge gained from this research could be used to improve processes, methods, tools, and security learning practices in OSS communities. This contribution is summarized and reported in RPs III and IV.

RQ 2: How can context-based approaches be applied in software security to motivate learners and improve learning outcomes?

This research question is addressed in the DC 2 of DSR activities, in which a novel method for software security instructional design and teaching is proposed and evaluated. The resultant contribution (C4) was published in RP IV. Contribution 4 concentrates on the ingredient of "high-quality teaching" by suggesting a novel approach for addressing the context in software security teaching and learning. Furthermore, it provides a practical demonstration of security learning material construction using the proposed context-based approach. This contribution points toward the ingredient of "willingness and effort on the part of the learner" in its focus on how context-based learning motivates learners and stimulates their interest in security learning.

RQ 3: How can one design an ontology that manages contextualized software security knowledge?

This research question, tackling current weaknesses in the security knowledge structure for effective learning, is investigated and answered in DC 3. The related contribution (C5) is reported in RP V, which concentrates on the ingredient of "high-quality teaching" by providing a concrete artifact for security knowledge modeling; instructors can use this tool to effectively deliver software security knowledge. Consequently, this contribution points toward the ingredient "opportunities to teach and learn" in its focus on providing knowledge resources.

RQ 4: How can one construct a learning system that facilitates context-based learning of security knowledge in software development?

A critical contribution of this thesis (C6) is the construction of a learning artifact (in DC 3)—a contextualized learning system—to provide more effective learning opportunities for software developers. Thus, the ingredient that C6 centers on is "opportunities to teach and learn"; it does so through designing and developing technical solutions for facilitating contextual software security learning and thereby encouraging more opportunities for security learning in software development. This contribution points toward the ingredients of "high-quality teaching" and "willingness and effort on the part of the learner" since the promising results offer evidence of the importance of context-based approaches in both pedagogical and software development environments.

Table 5.1 Summary of the contribution and research focuses

No	Contribution	Research paper	Primary Focus	Secondary Focus
C1	**Offered an overview of the security studies research literature related to OSS development.** • With a solid, systematic literature review, this thesis identifies and summarizes the strengths and weaknesses of security studies in the literature on OSS development from socio-technical perspectives. • This systematic review does not consist of a simple rearrangement of relevant data that is already known or that has been published. Rather, it was conducted according to a formal and controlled process. Thus, other professionals can follow the same protocol and judge the adequacy of the results. • This research work supplies researchers and practitioners with a firm basis for developing new security approaches for secure OSS development and addressing any of the identified limitations.	RP I	Social surround supportive of teaching and learning	Opportunity to teach and learn
C2	**Offered an empirical assessment of the security culture in OSS development environments.** • This research work fills the research gap by empirically investigating the social and cultural aspects of OSS security (a) to identify weaknesses and opportunities for improvement and (b) to illustrate progress in the security culture of OSS projects. • It provides practical insight regarding how to evaluate culture in software projects by operationalizing a security culture framework consisting of an assessment instrument. • By evaluating factors that would influence security in a positive way, this research work offers realistic and practical suggestions for software security managers working in OSS development. • This work provides empirical data and is significant regarding not only OSS project management but also successful collaboration models in software organizations or companies.	RP II	Social surround supportive of teaching and learning	Willingness and effort by the learner

(Continued)

Table 5.1: Continued.

C3	**Investigated knowledge acquisition and learning about software security in OSS development.** • This thorough investigation involved empirical examinations of OSS projects over time. Using the ethnographic method, this study explicated the rationalities of practice from an insider' s point of view; therefore, the emergent learning pattern was accurately identified. • This research work extends the spectrum of prior research on the effects of knowledge sharing and learning in OSS development at the individual and organizational levels by integrating socio-technical aspects of security learning in OSS communities. • It provides an in-depth examination of how cultural and social aspects complement each other in influencing security knowledge sharing and learning in OSS communities. • This research contributes to the body of knowledge in that it is the first such work to investigate the relationship among the security culture, organizational structure, and security knowledge sharing and learning in the OSS development context. This research confirmed that such a relationship exists and identified key factors influencing that relationship.	RP III and RP IV	Social surround supportive of teaching and learning	Good teaching
C4	**Designed, demonstrated, and evaluated a context-based approach to foster software security learning.** • This research suggests a new approach for addressing contextual aspects in software security teaching and learning. This new approach proposes that to encourage interest in security learning, educators should put knowledge in a context related to learners. By adopting this view, this research points to a new direction for context-based learning in security education. • The proposed novel learning approach features three concrete pillars: starting with a meaningful scenario, stimulating mental models for learning, and moving from concrete to abstract knowledge. • It describes a practical demonstration of the context-based learning approach to generating and presenting content knowledge with concrete learning materials; that demonstration produced sounder learning outcomes than conventional methods. • Since the proposed context-based approach enhances learning and contributes to improved software security education, the results can likely be generalized to other computer security topics (e.g., network security) and organizational settings (e.g., software development organizations).	RP V	Good teaching	Willingness and effort by the learner

(Continued)

Table 5.1: Continued.

C5	**Designed, demonstrated, and evaluated an ontology to improve the integration of contextualized and theoretical security knowledge.** • The research work provides a technical solution for managing software security knowledge in a structured manner that allows for flexible knowledge extraction using either contextual information or abstract security subjects. • The ontological model unifies security concepts and terminology; it is adaptable to the various contexts of software applications and generalizable to any computer security topic. The model can be applied in both pedagogical and software development environments. • The developed ontology differentiates itself from other research works since it models security knowledge with a diversity of software features and technologies, ensuring that knowledge users understand the security-relevant aspects of critical software features.	RP VI	Good teaching	Opportunity to teach and learn
C6	**Designed, demonstrated, and evaluated a contextualized learning system for software security that encourages learners to engage in learning tasks.** • The research work contributes to the practical demonstration of context-based learning in software security using technology-assisted learning environments, which integrates the proposed context-based artifacts (i.e., the learning approach and the ontological knowledge base). • The proposed learning system offers opportunities for learners to use their prior programming knowledge in software development to understand particular security elements. • This research work includes a comprehensive evaluation of the system's effectiveness in helping students achieve expected learning outcomes and learning satisfaction. This evaluation relied on the perspectives of potential users, including students and software developers. • The promising results of the experiments offer evidence of the importance of a context-based learning approach in both pedagogical and software development environments. • This research contributes to design science research and further demonstrates the validity of contextualized learning system development as a means of creating useful practical knowledge.	RP VII, RP VIII, and RP IX	Opportunity to teach and learn	Good teaching

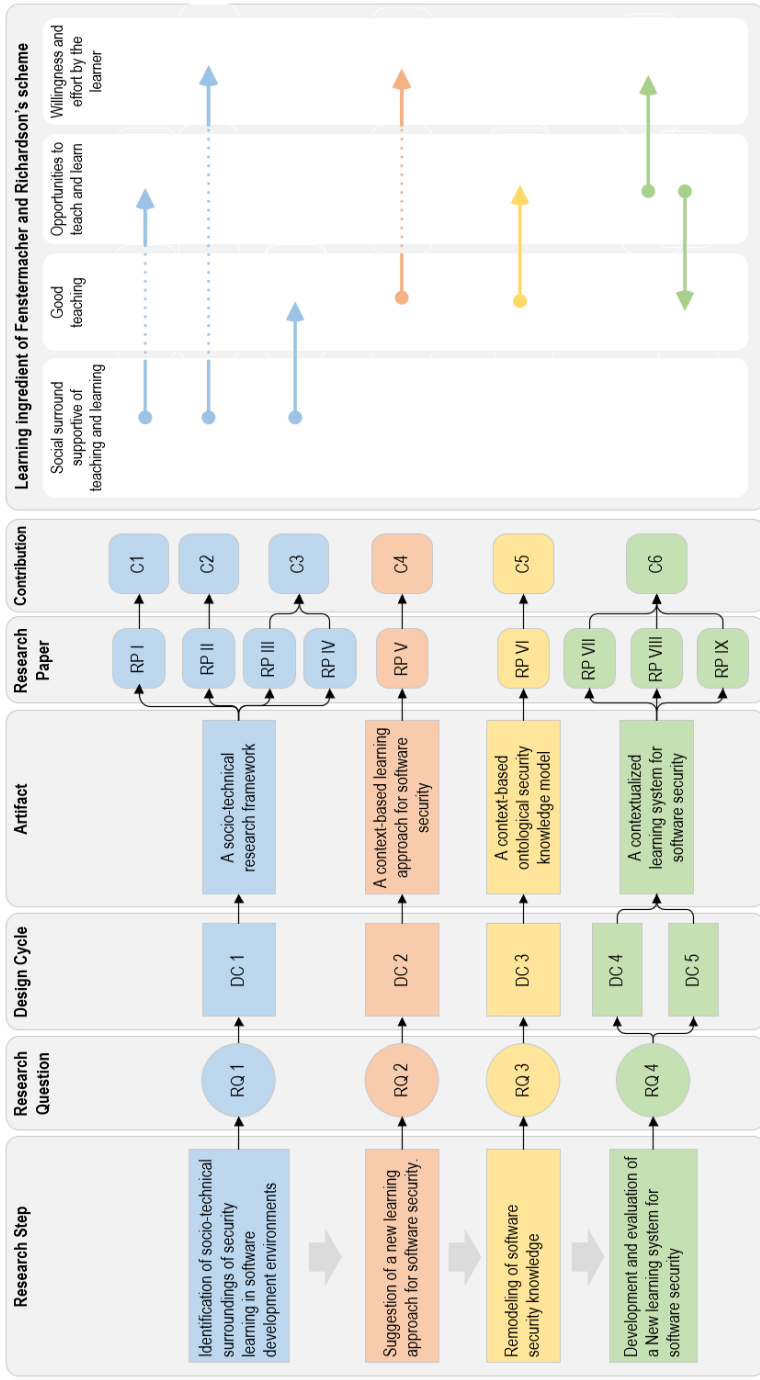


Figure 5.1: An integrated view of contributions in the thesis

Chapter 6

Conclusion

This chapter presents concluding arguments. The first two sections describe some limitations of the research and directions in which this research could be extended. The thesis then concludes with final remarks in the epilogue section.

6.1 Limitations of the Research

While this research has yielded some encouraging successes, identifying limitations is also important. The first limitation concerned with the extent to which the findings of a study can be generalized across different populations and contexts [68], that is, the socio-technical inquiry about developer learning of software security in OSS development (presented in RP II, III and IV). First, the number of subjects (OSS projects and developers) that participated in the empirical studies was rather small compared with today's enormous OSS projects and field workers. Second, while this research work focused on socio-technical aspects of security learning in OSS communities, some contextual characteristics of OSS projects were not included in the empirical studies while establishing the socio-technical research framework; examples are the size and maturity level of a project. [250]. Larger and more mature organizations may derive greater returns from knowledge sharing and learning because of their more substantial resources, and such organizations may also be more successful in reducing the learning curve [501]. Special attention should also be geared toward finding the human factors, which affect independent variables such as reputation, self-efficacy, and promotion. To provide useful results that can be generalized (or applied in different contexts), researchers must describe their results accurately and richly so that others can understand their relevance in a particular context. To this end, there is a need for further research efforts to improve the generalizability of this study to the entire OSS development phenomenon by considering a larger number of responses covering a range of diverse OSS projects,

and collecting more data for an analysis of potential differences based on unobserved heterogeneity in OSS development. Furthermore, the research would need to be repeated in other software development environments (e.g., proprietary software development) to justify (or falsify) the hypotheses in the framework to expand understanding toward security learning. The proposed socio-technical framework is useful as a "sensitizing device" [241] for allowing the development of more generic, social constructs that are useful in studying other social settings of software development.

The second limitation concerns the completeness and representativeness of the proposed context-based approach for security learning (presented in RP V). The strategies and methods were identified and synthesized through the literature review. The designed learning approach represents a subjective understanding of context-based learning in the domain of software security. Since this work is qualitative and based on the author's interpretation, the results might have been influenced by the author's culture and experiences. Extensive practices and iterations of review activities for the learning approach are therefore needed. Second, the experiment was conducted with a short-term study lasting approximately 40 minutes for each learning session, while the learning outcome was evaluated immediately afterward. Questions about the extent and duration of knowledge retention remain unanswered. To provide stronger shreds of evidence about the learning outcome, learning performance should be measured over a lengthy period.

The third limitation stems from the evaluation of the contextualized learning system (presented in RP VIII and RP IX). This artifact is a new and innovative product of this research. Although the learning system was examined through a two-phase evaluation process—a preliminary evaluation in the school setting (with bachelor students) and the final evaluation with OSS developers—the promising results may still be somewhat biased. First, the preliminary evaluation took place at a university. Students enrolling in the "Software Security" course were invited to freely take part in the experiment. It is by no means certain that those who chose to volunteer are representative of the population as a whole. The number of samples (36) also limited the generalizability of this study. Second, regarding the system evaluation in OSS development environments, the study adopted a questionnaire survey approach, with the findings based on self-reports from voluntary participants about their experiences with and perceptions of the proposed learning system. The issue here is whether the retrospective reporting of subjects accurately reflects reality. The results may not necessarily reflect how these individuals would interact with the system, the actual learning process, or the amount of time the individuals would spend engaging with the system. Moreover, the number of survey respondents was relatively low given the enormous number of OSS developers today. With that in mind, this work should be replicated with other sample populations to include more participants from diverse project settings. Further, more qualitative data collection techniques should be employed, such as focus groups, case studies, and in-depth interviews, to improve the accuracy of results and provide more evidence. For example, one can be more

confident in the results of surveys from interviews, thereby providing a more complete picture of the learning system.

6.2 Future Research Opportunities

Three key topics are introduced in this thesis: (a) contextual analysis of security learning in software development, (b) the context-based learning approach, and (c) the contextualized learning system. Several future research opportunities promise interesting results and insights regarding these areas, whether separately or jointly. These future opportunities are outlined below for each research area and the future extension of the learning system.

6.2.1 Contextual analysis of security learning in software development

This cross-disciplinary research on security learning in OSS development may serve as a foundation for continued integrative research on software development and follows the advice of Glass [165, 166]. The primary theoretical outcome of the research work is a conceptual framework for security learning in software development that allows users to view the real world in a certain way and that can contribute to increased conceptual clarity in discussions regarding security learning. As learning in software development is such a vital element of software development practice [213], there is a need to understand more about how software organizations can improve their capabilities for learning security knowledge. This thesis has offered insight into related challenges, but further research is needed to broaden the empirical foundation for analyzing and evaluating such improvement efforts. Further research could uncover more about the socio-technical relationship's role in supporting security learning; for example, researchers could explore how the relationship is mediated through practices such as group-based estimation and job rotation. Action research [125, 145] could be another effective strategy to underpin such an in-depth contextual investigation.

6.2.2 The Context-based learning approach

Context-based methods are still new and help to underpin software security education, which has not yet been the subject of in-depth research. Prior research has revealed that context-based science education helps students to more clearly see and appreciate the links between the scientific topics they are studying and their everyday lives; students' interest in and enjoyment of their lessons generally increase when they engage in context-based courses [365]. The innovation proposed in this thesis seems promising—and not only for the domain. In general, security education would benefit from research on context-based learning since it could represent a major element of the educational approach. The context-based approach distinguished in this thesis could be applied to study other educational fields, such as information

security studies and computer security. This application could lead to the discovery of more context-based learning and a broader description of the strategies involved in context-based approaches. Furthermore, studies on learning processes and factors potentially influencing learning processes are still needed to develop fine-grained models of context-based security learning. Future research could also examine the approach in combination with the proposed teaching strategies to pilot context-based learning at various points of the security instruction process, for example, the design of lectures and instructional materials. Evidence is lacking on how group work and school innovations affect long-term learning behavior dimensions. To perfect the approach, researchers could design specific strategies that are indicative of these dimensions and implement them over an entire school year. A more context-based curriculum that provides teachers with the necessary professional development will be feasible in the near future.

6.2.3 The contextualized learning system

This research yielded a concrete artifact designed to increase context-based learning of software security: the contextualized learning system. When this research is completed, the learning artifact should be carefully adjusted to incorporate other use cases for security learning so that knowledge users can acquire learning content according to their needs. For example, learning content can be provided according to the learner's knowledge level (novice, intermediate, or expert) or learning preferences. Furthermore, this research did not adopt socio-technical methods [8, 42, 250] to examine organizational cultural and structural effects while adopting this artifact in software development because such methods are typically used to examine the longer-term impact of new technologies on established learning practices. To fill this information gap, researchers could deploy the learning system in OSS development environments, providing the required security knowledge and observing the influence in terms of the security culture and software quality. If this new instrument is used, as intended, to support security learning in schools or software development projects, then this socio-technical level should be evaluated extensively. The above-described research possibilities could also support other potential extensions of the contextualized security learning system (e.g., the usability of user interfaces and the maintainability of security knowledge).

6.3 Epilogue

Software development is an ever-evolving field due to fast-paced product requirements, rapidly changing technologies, knowledge management, organizational structures and processes, and so on. The growing complexity of software development contributes to increasing challenges for developers in attaining the required security knowledge. To respond this challenge, this thesis offers a bird's-eye view of the state of the art of security learning in software development and provides a glimpse of what may lie ahead in the evolution of security, which includes

(a) elucidating a socio-technical approach for addressing real-world security learning problems and (b) highlighting promising directions and constructive thought around contemporary security education and learning themes.

First, this thesis highlights the complex relationship between technology factors and social factors and points to the need to address the socio-technical security learning gap between what organizations need to collaborate and what technology can provide in the context of software development. To that end, a socio-technical framework for security learning was developed based on a cross-disciplinary literature review and individual empirical studies. As the diversity of software systems and projects increases, there will no longer be a single approach (e.g., practices or tools) for achieving optimal security. On this aspect, organization should improve the integration of the activities for the learning of software security. This needs security expertise coordination and facilitation of security-knowledge transfer within the organization. It also requires peers' encouragement and support to interact, so that a positive culture towards security can be cultivated. The socio-technical conceptual framework provided in this thesis allows software organizations to think holistically about their strategies so that they can undertake the challenges through establishing a supportive security learning environment within the organization, consequently, helping developers strengthen their security competence. As this thesis argues, software developers face learning challenges of such magnitude that software organizations must take responsibility for ensuring that learning opportunities are continuously explored. Improving their capabilities to engage in security learning may enable software organizations to participate more mindfully in the so-called "Build Security In" initiative [294] and to thereby benefit more significantly from "learning in context" [363].

Second, contextualized teaching and learning represents a solution to problems in security education. This approach constitutes a step toward closing the increasing knowledge gap between the knowledge learned and the knowledge required by re-ordering the sequence of security knowledge to motivate learners. This thesis suggests that if security knowledge is taught in real-world situations that learners can connect to their real lives, learners will be able to recall the prior experience, resulting in their learning interest being aroused. Contextualized learning approaches can be powerful vehicles for shaping security learning in purposeful and interesting ways. Security educators may be encouraged by the concrete outcomes of this research, especially the finding that changing structures in security knowledge transfer facilitates fluid learning transitions—helping learners to arrive at security concepts when working in this context and to apply these concepts in various other settings. This finding implies, however, that new teaching practices are necessary for security education, which currently relies on conventional approaches.

Due to the complexity of software security, improving developers' knowledge to prepare them for this complexity is a challenging task. Considering the context is the key to reducing the gap between what developers know and what they need to know

about security. In this regard, educators and software communities must develop a learning environment in which context-based learning can be applied. This research's results regarding this context-based technique are promising. In evaluations, students cited the "learning journey" as a highly enjoyable and educational aspect of the course. Likewise, this research has received much positive feedback from software communities and industrial developers stating that they felt highly motivated to learn security knowledge using the contextualized learning system. While these results are positive, this research offers only an initial—albeit promising—a hint as to the potential of context-based support in security education and training for developers. In the future, we will keep promoting the context-based learning approach and the contextualized learning system in schools and industries. The accumulated security knowledge stored in this system can be utilized by learners from various disciplines and applies to a broad spectrum of public audiences.

Part II

Published Research Papers

Chapters and Corresponding Publications

This part of the thesis consists of the published research papers. The research papers and corresponding chapters in this part of the thesis are as follows:

Chapter 7:

RP I: Wen, Shao-Fang. "Software security in open source development: A systematic literature review." In 2017 21st Conference of Open Innovations Association (FRUCT), IEEE, 2017, pp. 364-373. doi: 10.23919/FRUCT.2017.8250205.

Chapter 8:

RP II: Wen, Shao-Fang, Mazaher Kianpour, and Stewart Kowalski. "An Empirical Study of Security Culture in Open Source Software Communities." 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE, 2019, pp. 863-870. doi: 10.1145/3341161.3343520.

Chapter 9:

RP III: Wen, Shao-Fang. "Learning secure programming in open source software communities: a socio-technical view." In Proceedings of the 6th International Conference on Information and Education Technology, ACM 2018, pp. 25-32. doi: 10.1145/3178158.3178202.

Chapter 10:

RP IV: Wen, Shao-Fang. "An Empirical Study on Security Knowledge Sharing and Learning in Open Source Software Communities." Computers, 2018, volume 7, issue 4. doi: 10.3390/computers7040049.

Chapter 11:

RP V: Wen, Shao-Fang and Katt, Basel. "Towards a Context-Based Approach for Software Security Learning." Journal of Applied Security Research. 2019, volume 14, issue 3, pp. 288-307. doi: 10.1080/19361610.2019.1585704.

Chapter 12:

RP VI: Wen, Shao-Fang and Katt, Basel. "Managing Software Security Knowledge in Context: An Ontology-Based Approach." Information 2018, volume 10, issue 6. doi: 10.3390/info10060216.

Chapter 13:

RP VII: Wen, Shao-Fang and Katt, Basel. "Development of Ontology-Based Software Security Learning System with Contextualized Learning Approaches." Journal of Advances in Information Technology. 2019, volume 10, no. 3, pp 81-90. doi: 10.12720/jait.10.3.81-90.

Chapter 14:

RP VIII: Wen, Shao-Fang and Katt, Basel. "Preliminary Evaluation of an Ontology-Based Contextualized Learning System for Software Security." In Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering. ACM, 2019, pp.90-99. doi: 10.1145/3319008.3319017.

Chapter 15:

RP IX: Wen, Shao-Fang and Katt, Basel. "Learning Software Security in Context: An Evaluation in Open Source Software Development Environment." In Proceedings of the 14th International Conference on Availability, Reliability, and Security. ACM, 2019, pp 58-67. doi: 10.1145/3339252.3340336.

Chapter 7

Software Security in Open Source Development: A Systematic Literature Review

Wen, Shao-Fang. "Software security in open source development: A systematic literature review." *21st Conference of Open Innovations Association (FRUCT)*, IEEE, 2017, pp. 364-373.

Abstract—Despite the security community's emphasis on the importance of building secure open source software (OSS), the number of new vulnerabilities found in OSS is increasing. In addition, software security is about the people that develop and use those applications and how their vulnerable behaviors can lead to exploitation. This leads to a need for reiteration of software security studies for OSS developments to understand the existing security practices and the security weakness among them. In this paper, a systematic review method with a socio-technical analysis approach is applied to identify, extract and analyze the security studies conducted in the context of open source development. The findings include: (1) System verification is the most cited security area in OSS research; (2) The socio-technical perspective has not gained much attention in this research area; and (3) No research has been conducted focusing on the aspects of security knowledge management in OSS development.

7.1 Introduction

It is indisputable that open source software (OSS) development has earned a key position standing in today's software engineering. Due to the uniqueness of the OSS model, the software security of OSS products has been widely discussed in security communities. However, the number of new vulnerabilities keeps increasing in today's OSS systems. According to the National Vulnerability Database (NVD), over 11,500 new vulnerabilities in OSS have been uncovered since 2012 [49]. These vulnerabilities open some of the most critical OSS projects to potential exploitation: Heartbleed and Logjam (in OpenSSL); Quadrooter (in Android); Glibc Vulnerability (in Linux servers and web frameworks); NetUSB (in Linux kernel), and many others [272, 357]. With increasing importance and complexity of OSS, the ineffective security practices to secure OSS development will result in more breaches that are serious in the future.

On the other hand, open source software is developed collectively by the online community of practices with a strong relationship between the technical and social interactions in a knowledge-intensive process. There are unique characteristics of OSS, such as community-based distributed development, volunteer workers, on-line information exchange, and informal integration of new contributors. These characteristics contribute to the high socio-technical complexity of OSS security, influence the applicability of software security practices in OSS development, and result in a need to manage the security practices and knowledge efficiently within the OSS communities. Moreover, the trustworthiness of the open-source depends on socio-technical aspects of the software security practices [106, 123, 302, 502], which include the expertise of the developers in the communities to produce secure code, quality of tools used in the development, the level of testing carried out before releasing the product, and the collaborative practices followed throughout the development cycle, etc. These aspects need a careful investigation from a socio-technical perspective as well [250].

Many studies have been conducted by both researchers and practitioners on the mechanisms of building security in OSS development. The overarching objective of this research is to summarize what we know about these security studies and to offer suggestions for research in OSS security. In this research, we carried out a systematic review of the existing literature to identify and classify the software security practices in securing the software products that are developed by the open-source communities. In addition, to investigate the security studies that are conducted in two aspects: socio-technical security and security knowledge management.

The rest of this paper is organized as follows. Section 7.2 describes the related work. The classification frameworks used in this SLR research is explained in section 7.3. The research method is explained in section 7.4. Section 7.5 describes each step in selection execution. In section 7.6, we give an overview of the literature review results.

Section 7.7 provides a discussion based on the result. Section 7.8 states the limitation of the study. Finally, we describe the conclusion in section 7.9.

7.2 Related work

In the open source research, there are few examples of the literature review. Hauge et al. [192] seek to identify how organizations adopt OSS. They classified the literature according to the ways of adopting OSS and evaluated the research on the adoption of OSS in organizations. Stol and Babar [428] aim to gain insights into the state of the practice of reporting empirical studies of OSS in order to identify the gaps to be filled for improving the quality of evidence being provided for OSS. Feller et al. [138] review 155 research papers to identify the kinds of open source project communities that have been researched and the kinds of research questions that have been asked.

In an introduction to a special issue, Scacchi et al. [391] provide an overview of the research on the development processes found in OSS projects. Crowston et al. [98] also present a quantitative summary of the literature of OSS development selected for the review and discuss findings of this literature categorized into issues pertaining to inputs, processes, emergent states, and outputs. Von Krogh and von Hippel [467] give an overview of some of the research on OSS and organize it into three categories: motivations of contributors, innovation processes, and competitive dynamics.

7.3 Classification framework

7.3.1 Software security areas

To identify the security practices in OSS development, we adopt the OWASP Software Assurance Maturity Model (SAMM) [72] as the guidance of the classification. The foundation of the model is built upon the core business functions of software development with security practices tied to each (see Figure 7.1). The building blocks of the model are the three maturity levels defined for each of the twelve security practices.

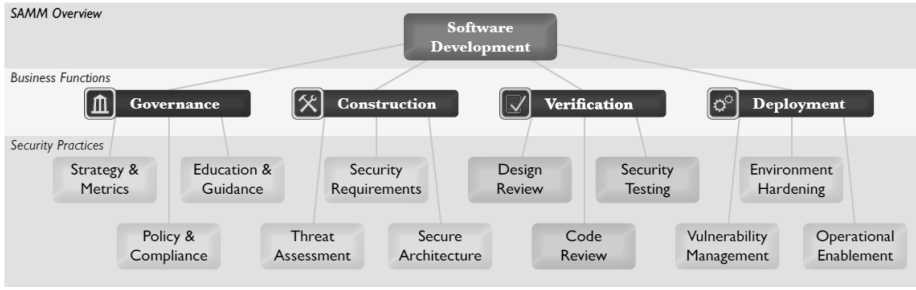


Figure 7.1: Software Assurance Maturity Model (Chandra [72])

7.3.2 Socio-technical perspectives

The software development process is not purely a technical task, but also a social process embedded within organizational and cultural structures [188]. The socio-technical perspective provides a deeper analysis of the relationship between the methods, techniques, tools, development environment and organizational structure [108, 109].

Our research is based on the Socio-Technical System (STS) and the Security-By-Consensus model (SBC) developed by Kowalski [250]. The STS model is depicted in Figure 7.2. This has two sub-systems include social aspects (culture and structures) and technical aspects (methods and machines). The SBC model is applied to define the detailed parts of the STS subsystem controls, illustrated in Figure 7.3.

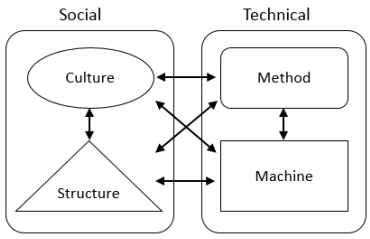


Figure 7.2: Socio-technical system (Kowalski [250], page 10)

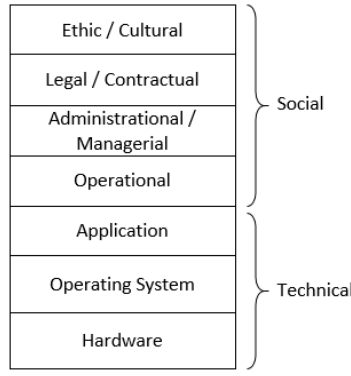


Figure 7.3: SBC Model (Kowalski [250], page 19)

7.4 Research Method

The design of this literate review is based on the original guidelines of systematic literature review provided by Kitchenham [239, 240] while also being guided by other systematic literature review articles in the area of open source software, such as Crowston et al.[98] and Hauge et al. [192]. The steps of the review include the definition of the research questions and the research protocol, conduct search for studies, screening of papers, data extraction, and data synthesis.

7.4.1 Research questions

This SLR aims to understand and summarize the empirical proofs as regards software security literature in the context of open source development. In addition, to investigate the security studies that are conducted in two aspects: socio-technical

security and security knowledge management. To achieve this aim, the research question addressed by our research is formulated as presented below:

RQ1: What research has been conducted on security practices and behaviors in the context of OSS development?
RQ2: What research has been conducted on the socio-technical security aspects associated with OSS development?
RQ3: What research has been conducted focusing on aspects of security knowledge management in OSS development?

7.4.2 Search strategy

The search strategy is used to search for primary studies including search strings and resources to be searched. The detailed description of the search strategies utilized in this research as explained below:

A Search term

To avoid overlooking relevant studies, all searches will be conducted using the combination of two categories of keywords in relation to "Open Source" (S_1) and "Security" (S_2), defined as follows:

- S_1 is a string made of keywords related open source, such *as "open source", "free software", "free/libre software", "OSS", "FOSS", "FLOSS".*
- S_2 is a string made up of keywords related to security, such as *"security", "secure", "insecure", "vulnerability", "virus", "malware", "exploits", "threat" and "hack".*

An example of a search done in the electronic data is described as follows:

"security" OR "secure" OR "insecure" OR "vulnerability") AND ("open source" OR "open-source" OR "free software" OR "free/libre software" OR "OSS" OR "FLOSS"

B Literature resources

Six primary electronic database resources were used to extract data for synchronizations in this research.

- ACM Digital Library (https://dl.acm.org).
- IEEExplore (http://ieeexplore.ieee.org).
- Springerlink (http://link.springer.com).
- Science Direct (http://www.sciencedirect.com).
- Scopus (https://www.scopus.com).

- Google Scholar (http://scholar.google.com/)

C Study Selection Criteria

The main inclusion criterion for this study is to include the software security studies that have been conducted in the context of open source development. The literature published during 2000-2016 is taken into consideration for the inclusion in search criteria. The detail inclusion criteria included are:

- Studies that describe security practices of OSS development.
- Studies that investigate security issues of OSS development.
- Studies that discuss the socio-technical characteristics of OSS security.
- Studies that discuss knowledge issues of OSS security.

Articles on the following criteria are excluded

- Papers that are not written in English.
- Studies that do not focus explicitly in OSS context, such as making use of OSS repositories as the study reference.
- Studies that only address OSS security concepts, such as comparing open source and proprietary (closed) software, and the use of OSS.
- Studies that focus on a specific open source platform or product.

7.5 Selection Execution

The search on the digital libraries initially identified 2942 papers. The selection execution was composed of four filter stages as shown in Figure 7.4. In stage 2, we individually reviewed the papers from the previous stage based on their titles and abstracts, and if necessary by skimming the full text and resulted in 167 papers. Next, in stage 3, to identify publications on security practices in OSS development, we individually went through the output of the second stage and evaluated the papers' topics by skimming the papers. Publications on the discussion of software security in the open source were included, while those do not focus explicitly on software security (only refer to software security as a side topic) and OSS context (only make use of OSS project data as the study reference) were rejected. Moreover, papers that focus on examining specific platform without contributing to OSS development were also excluded. Through stage 3, we discarded 74 of the 167 papers and selected 93 papers for further analysis.

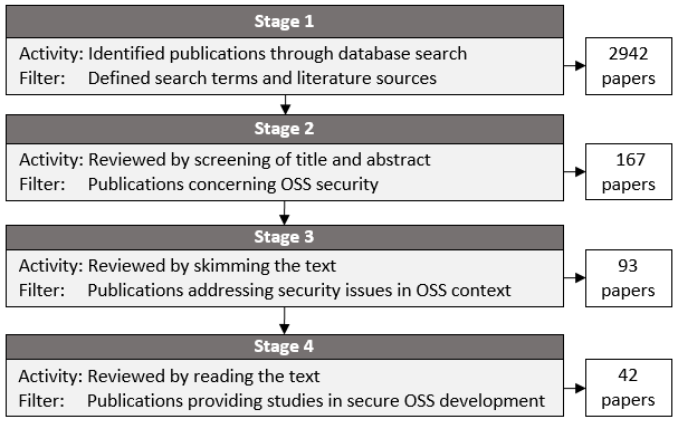


Figure 7.4: The paper screening process of SLR

Then we classified the publications from stage 3 into three categories: OSS concept where the authors discuss (debate) software security between open source and closed source, OSS adoption where authors present the security concerns in the use of OSS and OSS development. Of the 93 included papers, 27 were classified as open source concept papers, 24 as open source adoption paper, and 42 as OSS development papers. The OSS concept papers and OSS adoption papers may expand the understanding of OSS security issues but they are not providing any practical study to secure open source development. Hence, these papers were not included. Accordingly, the final stage of the review included 42 papers.

7.6 Result

This section presents an overview of the selected studies.

7.6.1 Publications by year

Table 7.6 (in Section 7.11 Appendix) shows the results of the research sources that have been found during SLR. Figure 7.5 illustrates the number of selected studies from the years 2000-2016. There are no significant studies related to our research topic in the year 2000 and 2001, and just a few papers were published between 2002 and 2005 (total of five papers in four years). This results from most studies of open source security in this period focus on the general discussion, such as concepts of open source security and debate on open vs. closed source security, etc. instead of security practices in open source development. The highest number of publications happened in the year 2014 (6 papers).

Table 7.1: Distribution of studies according to the publication venues

Type	Frequency	%
Conference Proceeds	29	70%
Journal	7	16%
Others (Book, Thesis, White paper)	6	14%

Table 7.2: Top five publication venues of identified articles

Source	Acronym	No.
International Conference on Open Source Systems	OSS	3
International Symposium on Empirical Software Engineering and Measurement	ESEM	3
International Symposium on Software Reliability Engineering	ISSRE	3
ACM Conference on Computer and Communications Security	ACM CCS	2
International Conference on Engineering and MIS	ICEMIS	2

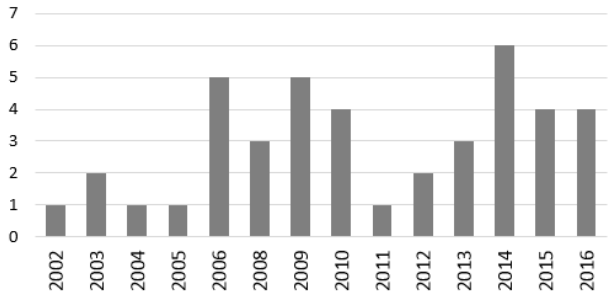


Figure 7.5: Number of publications versus the year

7.6.2 Publication venues and sources types

Table 7.1 presents the distribution of the studies' publication sources. Of the 42 studies, 70% (29 of them) were published in conferences, 16% (7 of them) in journals, 14% (6 of them) are distributed in books, thesis, and research white papers.

Table 7.2 presents the top five publication venues of some of the selected studies and the number of studies. Overall 34 publications venues are identified the cover different areas of computer science, such as software engineering, information system, and security, etc.; which means this study topic has received wide attention in the research community. One observation that can be made is that the leading publication venues are the type of conference proceedings, which are in the field of software engineering. This demonstrates the importance of OSS security research in software engineering and other related fields.

7.7 Discussion

This section describes and discusses the findings from the data extraction and analysis activities. The findings are presented in a graphical view and are organized by research question mentioned in section 7.4.1.

RQ1: What research has been conducted on security practices and behaviors in the context of OSS development?

Table 7.3 shows the categorization of security areas and related publications that fit the areas using OWASP SAMM presenting in section 7.3.1. Based on our review, the focus on OSS development varies in different papers. Figure 7.6 shows that 'Verification' is the most cited category in our SLR study (47%). This is due to the fact that open source development generally lacks formal system verification. The other reason is that vulnerabilities introduced in the design or construction stage will manifest themselves in code review or security testing if not detected earlier.

As shown in Figure 7.6, 'Construction' received the second-highest attention (29 %) in which the sub-category of 'Secure Architecture' has significantly higher numbers of studies (10 out of 14). The topics discussed in this area include the characteristics of security bugs [274, 433], vulnerable code change in OSS, [52, 54, 55], secure system design [87, 314, 383] and adoption of security tools [92, 225].

'Deployment' and 'Governance' are the two areas that receive the least attention in the research, 14% and 10 %, respectively. This may be due to open source projects do not typically have a corporate management staff to organize, lead, monitor, and improve the software development processes, which explains how hard the project management functions are in these two areas, such as strategic management, policy management, training, and operational enhancement, etc.

Table 7.3: Security areas of the selected studies

Category	Subcategory	Publications
Governance	Strategy & Metrics	[151, 253, 434, 504]
	Policy & Compliance	[504]
	Education & Guidance	n/a
Construction	Threat Assessment	[73]
	Security Requirement	[110, 274, 433]
	Secure Architecture	[52, 54, 55, 87, 92, 225, 274, 314, 383, 433]
Verification	Design Review	[141]
	Code Review	[1, 10, 13, 52, 53, 55, 126, 131, 299-301, 318]
	Security Testing	[92, 97, 179, 236, 306, 311, 355, 454, 470, 504]
Deployment	Vulnerability Management	[15, 17, 366, 372, 469]
	Environmental Hardening	[30]
	Operational Enhancement	[16]

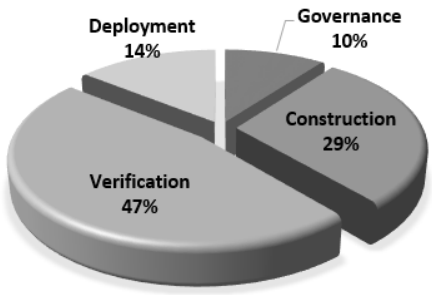


Figure 7.6: Frequency of studies in security areas

RQ2: What research has been conducted on the socio-technical security aspects associated with OSS development?

Our second focus is to investigate the socio-technical perspectives of OSS security revealed in these studies. Among the selected 42 studies, only two studies applied socio-technical approaches to address software security in the context of open source development [299, 372]: Study [299] proposed socio-technical metrics to describe the code review collaboration; study [372] analyzed socio-technical aspects of software problem management in OSS communities. Despite that, we performed a socio-technical analysis on these papers to understand what social and technical elements are highlighted in them, which was based on the socio-technical models mentioned in section 7.3.2. The analysis result is presented in Table 7.4.

From Figure 7.7, we see that the discussion of technical aspects has happened in 98% of the selected studies (41 out of 42). However, less than 50% of studies talked about the social-sector of OSS security (cultural, structural, legal, managerial and operational), and the average value is only 16%.

Looking at the information in more detailed, 'Operational' security has a higher frequency of discussion (45%, 19 papers). This is because the technical methods in software security are always accompanied by a certain process to have a successful implementation, especially at the working level. Compared with the significant portion of 'Operational' security, other social elements (cultural, structural, legal, and administration) of OSS security have not been given enough attention. They are noted in 7% (2 studies), 7% (2 studies), 2% (1 study) and 14% (7 studies) of selected studies, respectively.

RQ3: What research has been conducted focusing on aspects of security knowledge management in OSS development?

Table 7.4: Socio-technical aspects of the selected studies

Social-Technical Aspects		Publications
Cultural	An incentive of OSS participants	[504]
	Developer reputation	[53]
	Testing culture	[355]
Structural	Onion model vs. Source code maintenance	[87]
	Core-periphery structure vs. Code review outcome	[53]
	Distributed team vs. Developing a shared model in bug fixing	[97]
Legal	Governments policies	[504]
Managerial	Software repository management (Malware prevention)	[87]
	Risk analysis	[151]
	Coordination and communication mechanisms (Code review and security testing)	[53, 97, 306, 355]
Operational	Vulnerability handling behavior	[15, 16, 97]
	Secure design process	[73]
	Coding behaviors	[13, 52, 54, 274, 299, 301, 433]
	System testing behaviors	[355, 504]
	Security practices and tools adoption	[151, 225, 383]
	Code review behaviors	[53, 126]
	Quality assurance process	[306]
Technical	[1, 10, 13, 15-17, 30, 52-55, 73, 87, 92, 97, 110, 126, 131, 141, 151, 179, 225, 236, 274, 299-301, 306, 311, 314, 318, 355, 366, 372, 383, 433, 434, 454, 469, 470, 498, 504]	

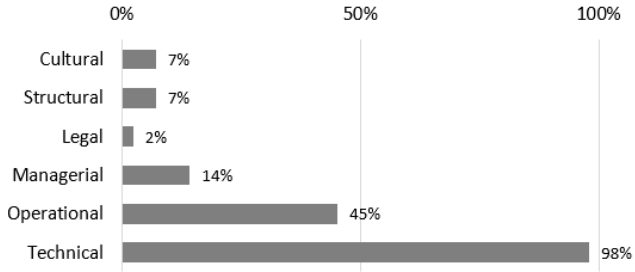


Figure 7.7: The coverage rate of socio-technical aspects

According to Table 7.3, there is no OSS security practice categorized in 'Education/Guideline' in which the security training and knowledge management are major activities. However, some papers did address knowledge problems in relation to OSS security, which is summarized in Table 7.5.

As we can see, the lack of security knowledge is the common problem that the research usually deals with. Among these papers, only [1] and [236] (2 out of 6) have proposed systematic solutions to tackle security knowledge issues, which aim to minimize the human efforts in software verification.

Table 7.5: Knowledge problems addressed in the selected security studies

Publication	Knowledge problems addressed in the study	Suggestions in the study
[1]	Lack of security knowledge in secure coding	Vulnerability prediction techniques can provide great help to OSS projects to deal with vulnerability flaws on a timely basis and with sufficient effort.
[236]	Lack of security knowledge in secure coding	Proposed an exploitable automatic verification system for secure open source software
[13]	Lack of security knowledge in secure coding	The OSS project should emphasize secure programming standards and reduce the use of unsafe statements.
[383]	Lack of knowledge in the adoption of security tactics	The OSS project should identify more practical security tactics and systematically incorporate them into the development process.
[52, 54]	There are differences among developers' knowledge and experience affect their likelihood of authoring vulnerable code change.	The OSS project should (a) create or adopt secure coding guidelines, (b) create a dedicated security review team, (c) ensure detailed comments during the review to help knowledge dissemination and (d) encourage developers to make small, incremental changes.

7.8 Limitation of the study

Even though this systematic literature review has been supported by a rigorous review methodology, a well-defined study protocol, and a close-knit paper screening process, it has some limitations.

7.8.1 Missing relevant publications

Our results depend on the used keywords and the limitations of the selected search engines. This approach misses the papers that are not indexed by the search engine and the papers that are not indexed with the keywords we used. We note that keywords are both discipline and language-specific and are not standardized. In order to limit the risk of incompleteness in keywords lists, we used alternative spellings and synonyms to build the search terms. Furthermore, by basing the search on a defined set of digital databases and the publication date, we excluded certain types of publications, work published through other channels or outside the defined timeframe. We can therefore not claim to have included all relevant publications. However, we adopted six popular digital databases with the full-text search to reduce the inherent limitations of search engines. We believe that our preliminary results cover the most relevant published literature.

7.8.2 Bias in the selection of relevant studies

Another potential limitation of the study is that subjective decisions can occur during the paper selection phases that cause bias in the selected execution. This is due to the lack of a clear description of the context, objective, and results of the selected studies. In order to mitigate this limitation, the selection process was carried out in an iterative way and the data extraction was realized. The selection execution in each paper screening stage was validated through an internal review process, which also helps to reduce the bias in the selection of studies.

7.9 Conclusion

This paper presents the systematic literature review that was conducted to identify open source studies with respect to the research practitioners for further work on open source security.

A total of 42 papers were selected in the SLR that met our inclusion criteria. The selected studies were analyzed and extracted data were classified into four main categories namely Governance, Construction, Verification, and Deployment. The result shows that security areas in Construction and Verification (Secure Architecture, Code Review, and Security Testing) are followed by researchers with more interests than other areas in Governance and Deployment.

Next, based on our research, the security studies in OSS development are mostly technology-driven. The socio-technical perspective has not gained much attention in this research area (2 out of 42 papers). According to the result of socio-technical analysis on the selected papers, the discussions between technical and social aspects seem quite unbalanced, either (Coverage rate: 98% versus 16% on average). The socio-technical perspective has as the main target to blend both the technical and the social systems in an organization. This can be viewed as a necessary condition within a security management framework as both aspects are of equal importance [152]. Technical security practice considering different social aspects (e.g., culture and structure) of open source development will assure the effectiveness and efficiency of the implementation of the tool.

Furthermore, the result of this SLR study also shows the gap that there is a lack of knowledge management aspects of open source security. Several researchers did mention the knowledge problems in securing OSS development, however, we cannot identify any study tackle this security issue from knowledge management perspectives.

Based on the finding of this research, we have come to the conclusion that the existing software security practices have limitations in supporting secure open source development. Secure architecture, code review, and security testing do help secure OSS products. However, as there is less research on socio-technical security aspects

and no discussion of security knowledge management in the context of OSS development, these practices, and software security knowledge cannot be effectively spread within the open source community. Since OSS participants are not experts on security in general and the domain knowledge of software security is vast and extensive, it is suggested that future research should explore socio-technical approaches in helping OSS developers learn the necessary security knowledge to fulfill the need of their work, further, to reinforce their behaviors towards OSS security.

The contribution of this work is to supply researchers with a summary of existing information about software security in open source development in a thorough manner, so as to provide a context in which to operate. It can also provide other researchers with a firm basis on which to develop new security approaches for open source development and address any of the identified limitations.

7.10 Acknowledgment

The author would like to thank Professor Dr. Stewart Kowalski and Professor Dr. Rune Hjelsvold of Faculty of Information Technology and Electrical Engineering at Norwegian University of Science and Technology, who have made comments and suggestions in this paper.

7.11 Appendix

Table 7.6: List of Selected Papers

Author	Year	Title	ID
Abunadi, I. & Alenezi, M.	2015	Towards cross-project vulnerability prediction in open source web applications	[1]
Alenezi, M. & Yasir, J.	2016	Open source web application security: A static analysis approach	[10]
Alnaeli, S. M., et al.	2016	On the Evolution of Mobile Computing Software Systems and C/C++ Vulnerable Code	[13]
Altinkemer, K. et al.	2008	Vulnerabilities and Patches of Open Source Software: An Empirical Study	[15]
Anbalagan, P. & Mladen V.	2010	Towards a Bayesian approach in modeling the disclosure of unique security faults in open source projects	[17]
Anbalagan, P. and Mladen V.	2008	Towards a Unifying Approach in Understanding Security Problems	[16]
Banday, M. T.	2011	Ensuring Authentication and Integrity of Open Source Software using Digital Signature	[30]
Bosu, A.	2014	Characteristics of the vulnerable code changes identified through peer code review	[52]
Bosu, A. & Jeffrey C. C.	2014	Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation	[53]

Bosu, A. et al.	2014	Identifying the characteristics of vulnerable code changes: An empirical study	[54]
Bosu, A. et al.	2014	When are OSS developers more likely to introduce vulnerable code changes? A case study	[55]
Chehrazi G. et al.	2016	The impact of security by design on the success of open source software	[73]
Colomina, I. et al.	2013	A study on practices against malware in free software projects	[87]
Cowan, C.	2003	Software Security for Open-Source Systems	[92]
Crowston, K. & Barbara S.	2008	Bug fixing practices within free/libre open source software development teams	[97]
Damiani, E. et al.	2009	OSS security certification	[110]
Edwards, N. & Liqun C.	2012	A Historical Examination of Open Source Releases and Their Vulnerabilities	[126]
Erturk, E.	2012	A Case Study in Open Source Software Security and Privacy	[131]
Feng, Q. et al.	2016	Towards an architecture-centric approach to security analysis	[141]
HP Fortify's Security Research Group	2008	How Are Open Source Development Communities Embracing Security Best Practices	[151]
Groven, A. K. et al	2010	Security measurements within the framework of quality assessment models for free/libre open source software	[179]
Jordan, T. B. et al.	2014	Designing Interventions to Persuade Software Developers to Adopt Security Tools	[225]
Kim, B. et al	2015	Design of exploitable automatic verification system for secure open source software	[236]
Krishnamurthy, S. & Arvind K. T.	2006	Bounty Programs in Free/Libre/Open Source Software	[253]
Li, Z. et al.	2006	Have things changed now?: An empirical study of bug characteristics in modern open source software	[274]
Meneely, A. et al.	2014	An Empirical Investigation of Socio-technical Code Review Metrics and Security Vulnerabilities	[299]
Meneely, A. & Laurie W.	2009	Secure open source collaboration: An empirical study of Linus' law	[300]
Meneely, A. and Laurie W.	2010	Strengthening the empirical analysis of the relationship between Linus' Law and software security	[301]
Martin, M. et al.	2005	Quality practices and problems in free software projects	[306]
Mockus, A. et al.	2002	Two case studies of open source software development: Apache and Mozilla	[311]
Mourad, A. et al.	2006	Security Hardening of Open Source Software	[314]
Nagy, C. & Spiros M.	2009	Static security analysis based on input-related software faults	[318]
Pham, R. et al.	2013	Creating a Shared Understanding of Testing Culture on a Social Coding Site	[355]
Ransbotham, S.	2010	An Empirical Analysis of Exploitation Attempts based on Vulnerabilities in Open Source Software	[367]
Ripoche, G. & Les G.	2003	Scalable automatic extraction of process models for understanding FOSS bug repair	[372]
Ryoo, J. et al.	2016	The Use of Security Tactics in Open Source Software Projects	[383]

Tan, L. et al.	2014	Bug characteristics in open source software	[433]
Tawileh, A. et al.	2006	Modeling the economics of free and open source software security	[434]
Vangaveeti, A.	2015	An Assessment of Security Problems in Open Source Software	[454]
Vouk, M. & Laurie W.	2013	Using software reliability models for security assessment - Verification of assumptions	[469]
Walden, J. et al	2009	Security of open source web applications	[470]
Xiong, M. et al.	2004	Perspectives on the Security of Open Source Software	[504]

Chapter 8

An Empirical Study of Security Culture in Open Source Software Communities

Wen, Shao-Fang, Mazaher Kianpour, and Stewart Kowalski. "An Empirical Study of Security Culture in Open Source Software Communities." *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2019, pp. 863-870.

Author Contributions— Initial conceptualization and framework of the research were developed by Shao-Fang Wen. The research design and methodology were reviewed by Stewart Kowalski. The manuscript was largely written by Shao-Fang Wen. Final paper review and editing were performed by Mazaher Kianpour.

Abstract—Open source software (OSS) is a core part of virtually all software applications today. Due to the rapidly growing impact of OSS on society and the economy, the security aspect has attracted researchers' attention to investigate this distinctive phenomenon. Traditionally, research on OSS security has often focused on technical aspects of software development. We argue that these aspects are important, however, technical security practice considering different social aspects of OSS development will assure the effectiveness and efficiency of the implementation of the tool. To mitigate this research gap, in this empirical study, we explore the current security culture in the OSS development phenomenon using a survey instrument with six evaluation dimensions: attitude, behavior, competency, subjective norms, governance, and communication. By exploring the current security culture in OSS communities, we can start to understand the influence of security on participants' security behaviors and decision-making, so that we can make realistic and practical suggestions. In this paper, we present the measurements of security culture adopted in the study and discuss corresponding security issues that need to be addressed in OSS communities.

8.1 Introduction

Open source software (OSS) is based on the principle that software programs should be shared freely among users, giving them the possibility of introducing implementations and modifications [168, 212]. OSS is released under license in compliance with the Open Source Definition as articulated by the Open Source Initiative (also known as the OSI). To create and sustain OSS, numerous technical and non-technical individuals interact with collaborating peers in online communities of practice [138, 140, 391]. The activities that these communities perform are usually called OSS projects. This development culture includes hundreds of thousands of distributed programmers voluntarily producing, sharing, and supporting their software with no monetary compensation for their efforts. Because of the low-cost software solutions, and the openness and real collaboration of the software development process, OSS has become an increasingly popular choice instead of closed source (proprietary) software: About 80% of companies run their operations on OSS [330], and 96% of applications utilize OSS as software components [50].

Due to the rapidly growing impact of OSS on society and the economy, the security aspect has attracted researchers' attention to investigate this distinctive phenomenon. As a result, numerous security practices for secure OSS development have been provided [481]. However, OSS vulnerabilities are being found at an increasing pace, nearly doubling from 2017 [420]. From a literature review of OSS security research using a socio-technical analysis, Wen [481] found that only 16% of papers talked about the social sectors of OSS security (cultural, structural, legal, managerial, and operational), and he concluded that existing software security practices have limitations in supporting secure OSS development. Because OSS in the socio-technical context is broader than the technical definition [390], technical security practices that consider different social aspects of OSS development will assure the effectiveness and efficiency of the implementation of the tool [481]. This can be viewed as a necessary condition within a security management framework, as the two aspects are equally important [152].

There is still a dearth of empirical research on the social study of OSS security. Thus, this study intended to complement the research by empirically investigating the social and cultural aspects of OSS security. As Zeitlyn [514] pointed out, we need to better understand the culture of the OSS movement and the corresponding social norms that regulate people's behavior. Culture has strongly influenced the formation of many security means in an organization, such as security policy, information ethics, security training, and privacy issues [395, 396]. Security culture can also support all organizational activities in such a way that security becomes a natural aspect of the daily activities of every individual [74]. By exploring the current security culture in OSS communities, we can start to understand the influence of security on participants' security behaviors and decision-making. Then we can evaluate what

changes would influence security in a positive way so that we can make realistic and practical suggestions.

The paper is organized as follows. After the introduction, in section 8.2, we present a review of the literature on OSS communities and security culture. In section 8.3, we present our research framework for this study. Section 8.4 describes the research methodology. We present the results of this study in section 8.5. In section 8.6, we discuss the results. We present the limitations of this study and the conclusion in sections 8.7 and 8.8, respectively.

8.2 Literature Review

8.2.1 OSS Communities

OSS is predominantly characterized by clan control, which is based on common values and beliefs [340], or clan- and self-governance [462], based on self-monitoring [237, 339]. In the OSS community, individuals interact with collaborating peers to solve a particular software problem and exchange ideas [123]. They work in geographically distinct locations of the world, and rarely or never meet face-to-face [280]. In OSS communities, social and technical interaction primarily occurs in a networked mediated computing environment populated with web browsers, a mailing list, a discussion forum, instant messaging programs, and other software development tools, such as version control systems, compilers, and bug tracking systems [390]. In this context, cooperation among members of OSS communities is maintained through an elaborate infrastructure that almost exclusively uses web technologies [212]. A strong culture and group behavior have been developed in connection with the community, enabled by the Internet [159].

The structure of OSS communities is fundamentally different from that of traditional project organizations of proprietary ("closed source") software development. Traditional software development projects tend to coordinate software development work through the organizational hierarchy and centralized planning [101], or they implement security control mechanisms, including behavior- or output-based control [335]. Unlike traditional organizations, OSS communities do not have a formal organizational structure, and projects in these communities are not dictated by formal plans, schedules, and deliverables [399, 406]. The organizational challenges faced by OSS development are considerable because the project must deal not only with the software engineering problems faced by a development team but also with the complexity of coordinating the efforts of a geographically distributed base of volunteers working on the software [322]. Moreover, proprietary software projects pay experts to come up with high-quality solutions, which is not necessarily true of open source projects, which rely on the motivation and personal interests of individual developers [447].

An OSS community has a unique structure depending on the nature of the system and its member population. In general, the initial OSS developer maintains a lead role and is responsible for the governance and coordination process [506]. The project leader, or the core team, usually partition the software development tasks into manageable modules and has participants choose what to work on according to their interests. The OSS development model allows developers to integrate with non-technical members to form a broader, more transparent community [463]. In this context, users and developers coexist in a community where the software grows and expands based on personal needs and benefits [178]. These benefits include fun, reputation, learning, enjoyment, and peer recognition [466]. Membership in the community is fluid; current members can leave the community, and new members can join at any time [406]. Consequently, individual ownership of products is not apparent in OSS communities; instead, recognition of expertise is important. Community members believe in shared risks, shared rewards, and shared ownership [505].

8.2.2 Security Culture

Security culture is the set of values, shared by everyone in an organization, which determines how people are expected to think about and approach security, and is essential to an effective personnel and people security regime [395]. Many researchers have defined security culture and identified its importance in organizations. Dhillon [118] defined security culture as "the whole of human attributes, such as behaviors, attitudes, and values which may contribute to the protection of all kinds of information within a certain organization." Schlienger and Teufel [396] defined security culture as "all socio-cultural measures that support technical activity methods, so that information security becomes a natural aspect in the daily activity of every employee." Martins and Eloff [288] defined security culture as the perceptions, attitudes, and assumptions that are accepted and encouraged by employees in an organization in relation to information security. Ngo et al. [325] suggested that security culture is the accepted behavior and actions of employees and the organization as a whole, as well as how things are done in relation to information security. In short, security culture is the way our minds are programmed to create different patterns of thinking, feeling, and actions for providing the security process [7].

Security culture covers social, cultural, and ethical measures to improve the security-relevant behavior of organizational members, and is considered a subculture of organizational culture [2]. This culture is recognized in the security community and scientific literature as one of the most important foundations of organizational security. Security culture is based on the interaction of people with information assets, and the security behavior they exhibit within the context of the organizational culture in the organization [105]. Security culture involves identifying the security-related ideas, beliefs, and values of the group, which shape and guide security-related behaviors [364]. The importance of creating a security culture within organizational

settings arises from the fact that the human dimension in information security is always considered the weakest link [289, 396, 451]. The results of numerous surveys suggest that people's attitudes and lack of awareness of security issues are among the most significant contributors to security incidents [154]. If appropriate security culture is neglected, individuals will not develop habitually secure behavior or take the initiative to make better decisions when problems arise. Therefore, the creation of a security culture is necessary for the effective management of information security.

8.3 Research Framework

In this section, we elaborate characteristics of a security culture that can be adopted in the context of OSS development. Based on a systematic review and a synthesis of relevant publications on security culture and information gathered from numerous pilot studies, we identified six dimensions of security culture: attitude, behavior, competency, subjective norms, governance, and communication.

8.3.1 Attitude

Attitude is an important factor that influences humans' emotions (how you feel or what you believe) and behavior [64]. Specifically, attitude can also refer to the degree to which a person has favorable or unfavorable feelings about an object [499]. The object can be an event, person, thing, place, idea, or activity. Other commonly used descriptions include behavior that is liked or disliked, desirable or undesirable, good or bad or behavior that is viewed positively or negatively [6]. Chia [80] asserted that in good security culture, individuals of the organization not only feel responsible but also have a sense of ownership about security. Unless they believe that security is important, people are unlikely to work securely, irrespective of how much they know about security requirements. Attitudes give a strong indication of individuals' disposition to act. For this study, attitude can be seen as OSS participants' feelings and emotions about the various activities that pertain to software security. Aspects include participants' belief (value) about security, responsibility for software security in the community, and positive thinking and perception of security requirements.

8.3.2 Behavior

The notion of behavior is based on what individuals do and relate to actual or intended activities [254]. According to Cox, Connolly, and Currall [93], human behavior is crucial in ensuring an efficient environment for information security. Essentially, security behavior is performed by individuals who are governed by instructions and requirements when using computer resources, but the way people think, believe, and subsequently, appreciate the organization of security affect how they behave [191]. Thus, security behavior can be seen as a function that frames the way that organizational actors collectively construct the meaning of different experiences of security tasks. The importance of participants' behavior in software

security management cannot be ignored. In the context of software development, security behavior includes the use of security technologies, adoption of secure coding practices, and compliance with organizations' security policies. Subsequently, risk-taking is another important component of security behavior, when the people involved in the design and/or operation of a system fail to perceive some set of conditions that might arise and cause the security of the system to be compromised [119]. People adjust their risk-taking behavior toward their "comfortable" level of risk (i.e., their "secure" level of risk).

8.3.3 Competency

Competency is defined as the underlying human characteristic that distinctly affects superior job performance in real-life and context-specific situations [28]. This characteristic is the collection of underlying knowledge and skills, which potentially enables some individuals to meet demands more effectively than others [69]. Competency, therefore, provides the potential capability to be skilled in relation to a specific goal or job task. To improve job performance and satisfaction, competency has been widely used to match employees to jobs by matching the competencies of a person to the job requirements [244], which causes individuals to feel that their behavior will not have any bad consequences. In the domain of software security, competency can be defined as software engineers' knowledge level and skills in protecting their software from a wide range of threats to software security, and with the ability to apply knowledge and skills productively (effectiveness). Having adequate competency regarding software security is a prerequisite to performing any software development task securely. Therefore, security competency may be regarded as an important factor to cultivate in security culture as the first line of defense in information security effectiveness.

8.3.4 Subjective Norms

A subjective norm is a person's belief about what people think about him or her should be done [135]. We recognize the term, subjective norms, as describing "directed normative relationships between participants in the context of an organization" [415]. Norms are a powerful means of regulating interactions among autonomous agents [26]. What is perceived as normal behavior in social settings has a strong influence on what is considered acceptable behavior in an organization, and what is not [456], independent of what the rules or formal policies dictate. Individuals are influenced by both—messages about expectations and the observed behavior of others [409]. For security culture, subjective norms represent a combination of perceived expectations of relevant individuals or groups along with intentions to comply with security-related tasks. It regards what is right and wrong regarding information security, involvement in organizational communication processes, and awareness of security policies. If the group considers information security an important and serious problem, then it is more likely that the individuals within that

group will value and follow the security policies. Conversely, if risk-taking is accepted within the group, then it is likely that greater risks will be taken. Failing to meet this expectation may incur a sanction against the offender. For example, members in OSS projects are often expected to follow a coding convention. Failure to adhere to this obligation may result in the code being rejected by the community. The level of intention toward a secure action is higher if the person has a positive attitude about and a subjective norm for the behavior [135].

8.3.5 Governance

Governance refers to the processes involved in developing and enforcing policies and norms for a given community or organization with the aim of structuring some set of activities [246]. Security governance is the means by which one controls and directs an organization's approach to security [246]. Security governance provides a framework in which the decisions made about security actions are aligned with the organization's overall business strategy and culture [107]. Thus, security governance is about decision making per se, which is concerned with setting directions, establishing standards and policies, and prioritizing investment and implementation. Effective security governance must provide mechanisms that enable managers to allocate expertise and responsibilities accordingly [480]. It requires roles and responsivities of security tasks, defined policies, implementation, and oversight mechanisms. In growing and maintaining an OSS project, people, such as the core contributors/maintainers, leaders, and community managers, must develop guidelines for writing and documenting code, implementing rules about licensing and distribution, determining methods for evaluating contributions to the project, and providing venues for like-minded users to communicate and build working, trust-based relationships (e.g., Slack channels and discussion forums) [410].

8.3.6 Communication

Communication, in simple terms, can be considered an interactive process of sending and receiving messages among individuals, groups, and organizations, including some form of feedback [248]. DeVito [116] defined communication as an act: "Communication refers to the act, by one or more persons, of sending and receiving messages that are distorted by noise, occur within a context, have some effect, and provide some opportunity for feedback." Clear, open, effective communication can create a sense of transparency in the organization, which builds trust between levels of employees. As Adams and Sasse [20] pointed out, insufficient communication with individuals in the organization "causes them to construct their own model of possible security threats and the importance of security and these are often wildly inaccurate." It is imperative that security has an internal voice in the form of broadcasting channels, ensuring policies, procedures, and relevant breaking news items are universally and regularly communicated. In the present study, communication refers to the methods OSS participants use to communicate security information within a

community, information transferring facilities, codification, and personalization information. Developers and users of an OSS project do not all necessarily work on the project in proximity. They require an electronic means of communication. Internet resources have the advantage of providing the community with an information infrastructure for sharing codification materials of software development in the form of hypertext, video, and software artifact content indexes or directories. Personalization communication has the inherent flexibility of transmitting tacit knowledge, and allowing for discussions and sharing interpretations that may lead to the development of new knowledge [51].

8.4 Research Methodology

This research adopted a quantitative approach to investigate the security culture in OSS communities. Quantitative research methods such as conducting surveys and the validation of research frameworks and questionnaires have been greatly applied in the information security discipline [398, 417]. Organizations can use survey instruments to study information security behavior in general [40]. The use of an OSS participant survey was deemed appropriate in this study, as the survey enables clear, direct, and objective answers to the questions presented to the respondents. For the purpose of this study, a self-administered web-based survey was used to collect individual-level perception data from participants in OSS projects.

8.4.1 Instruments

The survey instrument used in this study was the outcome of an iterative process of checking and refinement. We developed a questionnaire based on the six dimensions defined in section 8.3. The primary measurement items and the corresponding questions are summarized in Table 8.1. Some survey questions were inspired by existing studies, while others were created specifically to suit the research context of this study. Each item in the questionnaire was measured on a five-point Likert scale ranging from strongly disagree (1) to strongly agree (5).

8.4.2 Data Collection

Samples for the empirical study were randomly collected from participants in OSS development projects, available on GitHub. GitHub is an online database of OSS projects. Users and potential contributors can access information about projects, and download current versions of the software being developed. As in June 2018, GitHub reported more than 30 million users [164] and 57 million repositories [163], making it the largest host of source code in the world.

Table 8.1: Security culture dimensions and corresponding survey questions.

Dimension	Items	Question
Attitude	Value	• I believe software security is an important factor in achieving project success.
	Responsibility	• Software security is important to my work in software development.
	Positivity	• The requirements for software security do not interfere with my ability to get the job done.
Behavior	Acts	• I make the software components behave in a secure manner despite unexpected inputs or user actions.
	Compliance	• I adhere to the security principle and secure coding practices.
	Risk-Taking	• When I do my work, I assume that the software might be misused to reveal bugs that could be exploited maliciously.
Competency	Knowledge	• I know the principles and best practices for secure software development.
	Skills	• I can quickly identify specific coding errors or security vulnerabilities while examining the code base.
	Effectiveness	• I can apply methods or techniques adaptive to my project to prevent exploits against vulnerabilities.
Subjective Norms	Trust	• I believe the community can govern the security of software products.
	Supportiveness	• Members of the community help each other solve security issues.
	Expectation	• I am encouraged to work securely by members of the community.
Governance	Expertise	• There is a security team (or at least one member) who deals with software security for the project.
	Policy	• The project has a general policy for software security management (vulnerability reporting, security testing, etc.).
	Implementation	• The project has implemented secure coding practices (coding style, library, API, etc.).
Communication	Infrastructure	• There are dedicated communication channels (mailing list, forum, etc.) related to security subjects in the community.
	Codification	• It is easy for me to find specific security information in the community.
	Personalization	• I know where to go for advice related to a software security issue in the community.

The anonymous questionnaires were sent via e-mail to a list of OSS participants at the beginning of December 2017, and the data collection period lasted four months. Of the 321 questionnaires returned, 67 were excluded, because the respondents did not participate in an OSS community. In total, 254 respondent questionnaires were used for the final analysis.

8.5 Data Analysis

8.5.1 Respondent Demographics

Table 8.2 describes the general demographic information of the 254 respondents, in terms of gender, age, educational background. Nearly 90% of respondents were male, while there were only 9 female respondents. A large body of participants, that is 80%, was between 20 and 40 years old, and with a bachelor's degree (72.4%). Figure 8.1 shows the top 10 fields that the respondents' majors or anticipated majors. In the survey questionnaire, respondents were allowed to indicate more than one field if applicable. As the figure indicates, about 65% of respondents have been educated in the academic disciplines of computer and information sciences. In terms of characteristics of OSS communities, the largest group of seniority in the community was 47.6% of the total, with between 3 and 5 years of experience, and the 254 respondents were from various product profiles and horizons (Table 8.3).

Table 8.2: General demographic characteristics

Item	Category	Frequency	%
Gender	Male	228	89.8%
	Female	17	6.7%
	Prefer not to say	9	3.5%
Age	< 20	9	3.5%
	20–30	114	44.9%
	31–40	91	35.8%
	41–50	31	12.2%
	> 50	3	1.2%
	Prefer not to say	6	2.4%
Education	High school degree or equivalent (e.g. GED)	3	1.2%
	Associate degree (e.g. AA, AS)	12	4.7%
	Bachelor's degree (e.g. BA, BS)	184	72.4%
	Master's degree (e.g. MA, MS, MEd)	53	20.9%
	Professional degree (e.g. MD, DDS, DVM)	2	0.8%
	Doctorate (e.g. PhD, EdD)	3	1.2%

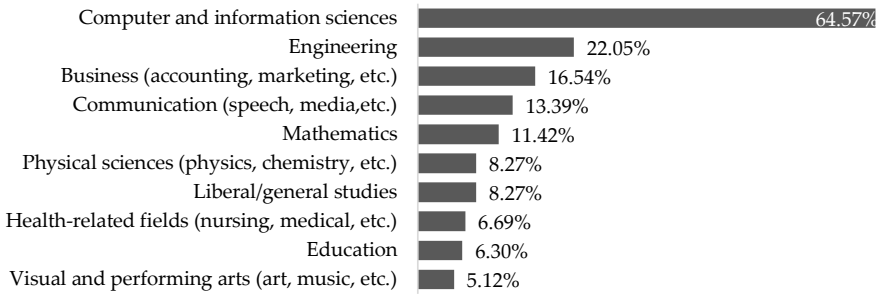


Figure 8.1: Top 10 fields that the respondents' majors or anticipated majors

Table 8.3: OSS Characteristics of the respondents

Item	Category	Frequency	%
Seniority in the community	< 6 months	4	1.6%
	6 months to 1 year	34	13.4%
	2–3 years	95	37.4%
	3–5 years	121	47.6%
	> 5 years	98	38.6%
Product Category	Browser, Content management	30	11.8%
	Database, File system	29	11.4%
	Security, Anti-virus, Encryption	24	9.4%
	Development framework	23	9.1%
	Education, knowledge management	19	7.5%
	Communication	19	7.5%
	Gaming, Entertainment	16	6.3%
	Healthcare	16	6.3%
	AI, Machine learning	13	5.1%
	Enterprise	12	4.7%
	Operating system	12	4.7%
	Word processing, Text editor	7	2.8%
	Retail & E-Commerce	7	2.8%
	Geospatial, Astronomy	5	2.0%
	Social media	4	1.6%
	Others	18	7.1%

8.5.2 An Overview of the Security Culture Scores

The mean scores of the security culture dimensions are plotted as a radar chart with six axes (Figure 8.2). As depicted in the chart, Attitude is the only dimension that reaches a mean value at the degree of 4.00. The respondents overwhelmingly reported a positive attitude toward software security. More concerning, however, is the evidence that a significant minority of respondents were unwilling or unable to put this positive attitude into practice. The mean value of participant-reported behavior is 3.90, showing that the behavior of OSS participants is at a mild level of maturity, but still, on average, insecure. The mean score for Competency is 3.72, indicating that, on average, the respondent communities faced moderate to serious in equipping relevant security knowledge and skills. The Subjective Norms aspects were not well developed, as the mean score was 3.74. Notably, this study revealed there was very weak security governance to support security culture (mean = 3.37), suggesting that an insufficient complement to security expertise, as well as limited establishment and implementation of security policies. Last, Communication of security information in the OSS communities studied was, on average, very weak (mean = 3.28). Communication is the least developed dimension in security culture, as the mean is the lowest of all six dimensions.

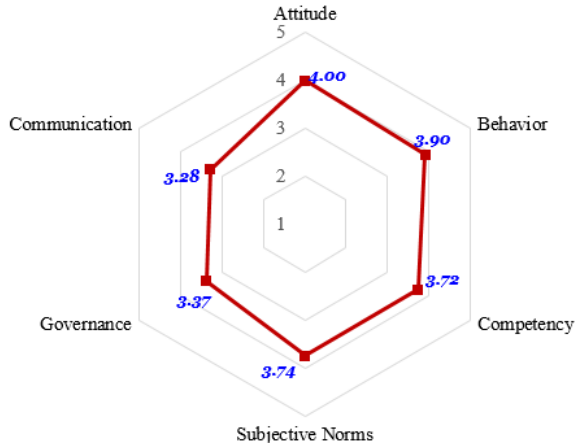


Figure 8.2: The mean score of security culture dimensions

8.5.3 Attitude

The results (Table 8.4) show that the vast majority of respondents (90%) held the value that security was an important factor in achieving project success. This could be a result of high-profile vulnerabilities and security incidents of OSS in recent years, which have generated a lot of adverse publicity for OSS development. Despite acknowledging the value of security for the project, only 56% of respondents agreed that software security was important to their work in the community, and a quarter of the survey population held a neutral position while answering this question. In addition, the mean score was statistically significantly low (3.67) in this dimension. OSS participants were still skeptical about the obligation to "build security in," as part of their jobs or roles. They had an inadequate understanding of how individual actions contribute to the security of the software system as a whole. In addition, we found that a third of respondents (disagree and neutral) felt security might interfere with their ability to get the job done. The result indicated that OSS participants viewed security as something that was necessary to their projects, but at times, also expressed their perception of the conflict between the security requirements and how they were used to writing code. Thus, the respondents shifted responsibility for software security to the community or public.

Table 8.4: Descriptive analysis of the Attitude dimension

	Frequency (Percentage)					
Item	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean
Value	2(1%)	8(3%)	16(6%)	76(30%)	152(60%)	4.45
Responsibility	11(4%)	25(10%)	76(30%)	67(26%)	75(30%)	3.67
Positivity	12(5%)	22(9%)	54(20%)	65(26%)	101(40%)	3.87

8.5.4 Behavior

We found that most respondents agreed about secure coding behavior. As the results reveal in Table 8.5, 70% of respondents agreed with the following statement about security acts: "I make the software components behave in a secure manner despite unexpected inputs or user actions." Similarly, nearly three out of four (74%) reported that they complied with secure coding policies in their work. However, in the two questions, the proportion of neutral responses was relatively high (20% and 17%, respectively). In addition, a minority group (nearly 10%) actively disagreed with the two statements about secure acts and compliance. The two groups of people (neutral and disagree) totaled nearly one-third of the survey population, which presents notable issues for OSS security. Most OSS participants might primarily focus on their immediate goals that usually involve functional requirements and performance, instead of security. In addition, the further result showed 38% of respondents performed risky behavior at a certain level in secure software development. They were likely to skip policies or bypass them to make their job easier, unaware of the potential damage, thinking that attackers would not be interested in their applications, or that their company was not big enough to be a target for attacks.

Table 8.5: Descriptive analysis of the Behavior dimension

	Frequency (Percentage)					
Item	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean
Acts	5(2%)	19(7%)	51(20%)	94(37%)	85(33%)	3.93
Compliance	4(2%)	18(7%)	44(17%)	96(38%)	92(36%)	4.00
Risk-Taking	6(2%)	25(10%)	66(26%)	80(31%)	77(30%)	3.78

8.5.5 Competency

Worryingly, fewer than two-thirds of respondents, that is, 66% (in Table 8.6), said they had knowledge about general principles and best practices for secure software development, and only 65% said they had relevant skills for identifying specific security errors in code repositories. In addition, more than one-third of respondents (34%) did not agree with the following statement: "I can apply methods or techniques that adapt to my project to prevent exploits against vulnerabilities." The issues of OSS participants' lack of security competency mostly resulted from the fact that they come from various academic disciplines (as shown in Figure 8.1), and might not have formal college-level security training. Thus, a lot of confusion remained in participants' minds about what was secure code and what the project wanted. This confusion forced them to take risks based only on their personal experience, without fully considering the project's requirements and priorities.

Table 8.6: Descriptive analysis of the Competency dimension

	Frequency (Percentage)					
Item	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean
Knowledge	9(4%)	23(9%)	55(22%)	102(40%)	65(26%)	3.75
Skills	7(3%)	28(11%)	54(21%)	103(41%)	62(24%)	3.73
Effectiveness	12(5%)	26(10%)	48(19%)	110(43%)	58(23%)	3.69

8.5.6 Subjective Norms

The degree to which OSS participants trusted their community in the governance of software security was relatively high in the dimension of Subjective Norms. Nearly 80% of respondents conveyed their trust of their communities' security governance (Table 8.7). This result implies that OSS projects relied on the communities' management and control, and are conducted to a great degree to ensure the security protocols are carried out. However, only 65% agreed with the statement, "Members help each other solve security issues." Normative support for security tasks was not clearly perceived among OSS participants. In line with this, it perhaps is not surprising that only 51% thought that they received encouragement and expectation from their peers to work securely in OSS communities, while more than 20% did not agree that they had been influenced by other members regarding secure software development. The OSS participants did not perceive strong norms in their communities, something that could promote and reward behavior that serves the security quality of their software products.

Table 8.7: Descriptive analysis of the Subjective Norms dimension

	Frequency (Percentage)					
Item	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean
Trust	8(3%)	19(7%)	28(11%)	95(37%)	104(41%)	4.06
Supportiveness	15(6%)	23(9%)	51(20%)	84(33%)	81(32%)	3.76
Expectation	24(9%)	31(12%)	69(27%)	76(30%)	54(21%)	3.41

8.5.7 Governance

Regarding the complement of security expertise in OSS communities, less than half of the survey population (46%, Table 8.8) clearly reported that there were security teams (or at least one person) dealing with software security in their communities, implying that a considerable portion of participant communities (54%) did not possess sufficient expertise to fully address complex security risks. OSS projects do not usually have the monetary resources in software security that companies producing proprietary software have. The people hosting the project have to do it in their spare time, making the level and motivation of security conduct questionable. This situation could also result in fewer security policies and a low implementation

rate for secure practices in OSS development. In this study, security governance in OSS communities was either weak or problematic, as only half of the respondents (51%) agreed with the statements about the situations in the two measurement items, policies, and implementation.

Table 8.8: Descriptive analysis of the Governance dimension

	Frequency (Percentage)					
Item	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean
Expertise	21(8%)	35(14%)	83(33%)	68(27%)	47(19%)	3.33
Policies	25(10%)	44(17%)	56(22%)	71(28%)	58(23%)	3.37
Implementation	21(8%)	47(19%)	57(22%)	65(26%)	64(25%)	3.41

8.5.8 Communication

Only 41% of respondents reported that dedicated communication channels related to security subjects existed in the community (Table 8.9). We found that only 35% of participants agreed with the statement, "It is easy for me to find specific security information in the community," and nearly 40% disagreed. OSS projects normally publish their own coding guidelines, a set of conventions (sometimes arbitrary) about how to write code for that project. However, OSS projects rarely address the security requirements in documentation to help drive the team to understand the prioritized security needs of the entire project. Thus, newcomers might feel that comprehending security requirements from exploring the website is hopeless; thus, they prefer to start with programming. In contrast to striving for codified security information, respondents felt at ease in asking for guidance or recommendations using available communication channels in their communities. Nearly 70% of respondents said they knew where to go for advice about security for their personal needs.

Table 8.9: Descriptive analysis of the Communication dimension

	Frequency (Percentage)					
Item	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean
Infrastructure	34(13%)	65(26%)	51(20%)	57(22%)	47(19%)	3.07
Codification	27(11%)	69(27%)	68(27%)	54(21%)	36(14%)	3.01
Personalization	17(7%)	28(11%)	34(13%)	96(38%)	79(31%)	3.76

8.6 Discussion

We identified that a key inhibitor of OSS culture is the "it's not my responsibility" attitude. The survey data in Table 8.4 showed that there was a strong reliance on participants' mindset on other methods (members, processes, and technology) to take care of software security. The lack of responsibility could occur when security is not considered part of a developer's everyday duties, or when developers expect security

is handled elsewhere, such as by the core team or other community members. Given the openness and freedom of OSS, it is not surprising that OSS developers ease their workload by passing the responsibility for software security to others when possible. As long as developers are not held responsible for security tasks related to their code, they would rather spend their time on aspects for which they will be held responsible.

Developers want to write more secure code, but this might not be a priority for their work. They focus on contributing software code with the perception to become good application developers, but not necessarily security experts. Getting code out quickly, albeit with vulnerabilities that they discover and fix later, maybe a better fit with their personal goals. As the analysis results are shown for the measurement of Positivity (Table 8.4), it is not that OSS developers do not want to develop secure products, but they are more interested in delivering new functionality to increase the features of their software products. Furthermore, in the aspect of risk-taking, depicted in Table 8.5, OSS participants might think that hackers are not interested in their applications, or that they are not famous enough to be a target for attacks. Thus, they see no perceived risk, and security efforts lack value. This perhaps indicates that OSS communities still have some way to go in ensuring that software security is high on the list of project priorities, and gets participants' attention in promoting positive security best practice.

In addition to the lack of incentives to focus on strengthening security, our study revealed a missed set of means in terms of security practice reinforcement and demonstrates a clear knowledge gap that must be addressed by OSS communities. The data analysis in Table 8.6 indicated that two-thirds of respondents evaluated themselves as equipped with security competency, but the other one third did not. Still, OSS developers today are, most likely, unaware of the many ways they can introduce security problems into their code, and do not have the wherewithal to fix them when they are found. In view of the gap in the skills and knowledge necessary for secure OSS development, the lack of appropriate security competencies is clear. OSS developers or other participants do not traditionally receive formal education or training about software security [483]. Programmers make security errors because they are unaware that their code will be attacked, and have no knowledge of methods by which their code can be secured. Knowledge is not the motive for human information security behavior; however, the lack of knowledge is a barrier to developing the desired behavior [233]. We believe a closer look at security training also seems to be needed.

To effectively deal with security problems, OSS participants need greater awareness of specific errors in the context of their own development. Thus, security knowledge transfer within the OSS community is required to help them know about the threat to their own products, so they are motivated to respond. They also need to know what it means to write secure code, and how to find and correct the errors that cause security flaws. Improving participants' competence in security can improve their confidence when a user is placed in the adverse condition of using the software [408].

It also makes the participants feel that their behavior will not have any bad consequences. With respondents' broadly positive attitude to security, OSS communities clearly need to place more focus on providing members with information related to security subjects, offering opportunities for learning and supporting self-development of security knowledge.

However, based on the analysis of subjective norms in Table 8.7, weak subjective norms support security culture in OSS communities. Only half of the respondents thought that they were encouraged by community members in terms of secure software development. Thus, OSS communities should enforce adherence to the mutual norm of security aspects, making cooperation between developers a goal, as well as part of the success of the project. Research indicated that in teams where security was part of the organizational culture and support for security tasks was available, individuals were more motivated to focus on security [303, 418]. This could be because they are confident in performing their security tasks, especially when they feel support from peers. This behavior could result in a snowball effect and lead to motivating more community members to recognize the importance of considering security as their peers do.

Developers do not like to feel exploited. If they believe that the other members of the project will not contribute equally, the norm of reciprocity is violated [37]. In the context of OSS development, peers' positive encouragement or expectation of secure coding behavior could increase developers' bonds with their teams, for example, with the feeling that they see the value of the community, and thus, perform the expected behavior for the team. As a result, rather than performing security tasks purely to follow an order or commission, the participants internalized such work, accepted it, and experienced a willingness to act. This internalization of security has a statistically significant positive effect on persistence and performance [382]. We believe that OSS communities will greatly benefit from a security culture where an individual takes more responsibility for the security of the collective he or she is a part of and is assured help if he or she encounters security crimes.

This study also exposes a problem that there was very weak security governance to support security culture. OSS communities differ from common enterprises in their coordination and organizational structure. The work is done on a voluntary basis, and there are fewer guidelines regarding time and intensity of work. Software security should not only be the domain of the core developers. On the one hand, those responsible for core development tasks must understand the importance of the scope of software function protection. On the other hand, participants must be informed of the general process and methods to provide protection during the entire software development cycle. In this regard, OSS communities can utilize a security team or experts to define security requirements and best practices, help perform code reviews, and provides the necessary security knowledge for the software development staff. The team acts as the known point of escalation for security issues encountered by developers if local champions cannot resolve them. It is also responsible for

sympathetically setting standards or practices, as developer members will have a working knowledge of how security practices are best implemented.

To design functional and effective security governance, OSS projects must not only be responsible for security expertise coordination but have the ability to execute corresponding security policies. Security policies or guidelines have to be readily accessible or available to participants to ensure that they will not be ignored. Therefore, OSS communities must have the ability to convey the criticality of maintaining security to the whole project team. However, as this study reveals, Communication gains the lowest score among the six dimensions of security culture in OSS communities. To overcome the communication problems, OSS communities need to provide a communication strategy to ensure that participants have reached security information, the codification knowledge when they need it, and importantly, are aware of where they can locate it. For example, specific security web pages can be included in the project website or repository, serving as an information clearinghouse. With just a glance, participants understand they need to pay attention and take any recommended action immediately. Through this structural mechanism, the security knowledge gains valuable insights from the community, and further, facilitating discussion and decision making and sharpening personalization knowledge.

8.7 Limitations

Several limitations of this study should be noted. First, the survey relied heavily on self-reported data from participants about their perceptions and activities in secure OSS development. Respondents may have wanted to portray an ideal image of their security attitude, behavior, or knowledge within the workplace, rather than reality. Although participants were not required to name their project and were given assurances of anonymity, respondents may still have reticent in reporting their actual behaviors. Second, the samples were chosen opportunistically from GitHub repositories, and the number of responses obtained from the survey was small compared with the enormous number of OSS projects and field workers today. Thus, there is a need for further research efforts focused on accumulating more evidence that is empirical, and data to break through the limitations. These efforts should improve the generalizability of this study to the entire OSS development phenomenon, by considering a larger number of responses covering a range of diverse OSS projects.

8.8 Conclusion

In this paper, we present a security cultural analysis in the context of OSS development. Measurements of security culture and the corresponding issues that must be addressed in OSS communities were defined and discussed. OSS is a core part of virtually all software applications today. The number of OSS projects has

increased significantly over the last 5 years [420]. It is easier than it has ever been to create a new OSS project, as well as use other projects from other members of the community. The barrier to entry has decreased so that a large number of enthusiastic amateur developers build a variety of apps and share their code in their spare time. This diversity of OSS projects is fantastic, but there is a shortage of developers entering the profession with software security expertise. With the increasing speed of development and sharing, convincing developers of the importance of security is challenging. Previously, OSS projects were focused on functionality and speed to market as their main goals. However, under pressure from a rising number of malicious threats and with tighter privacy protection laws coming into force, OSS communities have had to rethink their priorities. As the diversity of OSS products and projects increases, there will no longer be a single approach (e.g., practice, tool, heroic effort, or checklist) for achieving an optimal security culture suited to all communities. We believe that every technology developer has a responsibility to implement and participate in such a process. This is fundamental to achieving a security culture in a software organization. Furthermore, OSS communities should establish rules and norms, roles, and methods, that is, to cultivate and maintain a culture that values positive security attitudes and behaviors.

Chapter 9

Learning Secure Programming in Open Source Software Communities: A Socio-Technical View

Wen, Shao-Fang. "Learning secure programming in open source software communities: a socio-technical view." *Proceedings of the 6th International Conference on Information and Education Technology*, ACM 2018, pp. 25-32.

Abstract—In open source software (OSS) communities, volunteers collaborate and integrate expertise to develop the software online via the Internet in a decentralized, highly interactive and knowledge-intensive process. The development of qualified and secured software products relies mainly on the ability of OSS participants to acquire, refine and use new aspects of secure programming knowledge. Many OSS proponents believe that open source innovation offers significant learning opportunities from its best practices. However, studies that specifically explore the learning of software security in the context of open source development are scarce. This paper aims to empirically assess present knowledge sharing and learning about secure programming knowledge in the context of OSS communities utilized a socio-technical approach on OSS projects based on an ethnographic observation. Our motivation is not only to evaluate the knowledge sharing and learning mechanisms and the extent to which they may be viable and successful but also to gain insight into the security culture and project factors that affect learning processes of secure programming in OSS communities.

9.1 Introduction

Open source software (OSS) is based on the principle that computer programs should be shared freely among users, giving them the possibility of introducing improvements and modifications. OSS is at the core of today's IT infrastructure and information systems: about 80% of companies run their operations on OSS [330] and 96% of applications utilize OSS as the software components [50]. OSS security has been the focus of the security community and practitioners over the past decades. Many studies have been conducted by both researchers and practitioners on the mechanisms of building security in OSS development [481]. However, the number of new vulnerabilities keeps increasing in today's OSS systems. The Blackduck 2017 Open Source Security and Risk Analysis report has announced that 3623 new OSS vulnerabilities occurred in 2016 – almost 10 per day on average and a 10% increase from 2015 [50]. According to the 2017 NIST[36] report, about 67% of vulnerabilities are due to programming errors; the rest are due to configuration or design problems [256]. In particular, a strong majority has been found to be classic errors that are fairly well known, like buffer overflows, cross-site scripting and injection flaws. With today's increasing importance and complexity of OSS, the ineffective learning of knowledge and skills relevant to secure programming practices in OSS development will result in more breaches that are serious in the future.

Learning secure programming is a difficult and challenging task since the domain is quite contexted specific, and the real project situation is necessary to apply the security concepts within the specific system. In the context of OSS, the development of qualified and secured software products relies mainly on the ability of developers to acquire, refine and use new aspects of secure programming knowledge in their communities. Many OSS proponents believe that the OSS community offers significant learning opportunities from its best-practices [204, 257], which are different from the education of the traditional model [71, 144]. However, studies that specifically explore the learning of secure programming in OSS communities are scarce.

On the other hand, OSS is developed collectively by the online community of practices with a strong relationship between the technical and social interactions in a knowledge-intensive process [198, 245]. As Scacchi [390] points out, the meaning of open source in the socio-technical context is broader than its technical definition and includes communities of practice, social culture, technical practices, processes, and organizational structures. This can be viewed as a necessary condition within a learning framework as both aspects are of equal importance [29]. Technical learning mechanisms considering different social aspects (e.g., organizational culture and structure) of OSS development will assure the effectiveness and efficiency of the learning process.

[36] The National Institute of Standards and Technology (NIST) is a physical sciences laboratory, and a non-regulatory agency of the United States Department of Commerce (https://www.nist.gov/).

Given the background, this study was designed to empirically assess present knowledge acquisition and learning about secure programming in OSS communities utilized a socio-technical approach on OSS projects based on an ethnographic observation. In contrast to earlier researchers, which have focused on generic learning in OSS communities, our study aimed to observe OSS participants' perception of learning about secure programming knowledge. Our motivation for this study is not only to evaluate the knowledge sharing and learning mechanisms and the extent to which they may be viable and successful but also to gain insight into the security culture and project factors that affect learning processes of secure programming in OSS communities. The rest of this paper is structured as follows. Section 9.2 describes the literature review, including learning in open source communities and the views on socio-technical aspects. The research method is explained in Section 9.3. In section 9.4, we present the result of data analysis. Section 9.5 provides a discussion based on the result. Section 9.6 states the limitation of this study. Finally, we describe the conclusion in section 9.7.

9.2 Literature Review

9.2.1 Learning in Open Source Communities

In open source software (OSS) communities, volunteers collaborate and integrate expertise to develop applications and solve particular programming problems via the Internet in a decentralized, highly interactive and knowledge-intensive process [198, 245]. Larger numbers of technical and non-technical users get participation in activities that are essential for the OSS development process, as well as the maintenance and diffusion of the software [138, 140, 391]. The activities that these communities perform are usually called OSS projects, in which the software source code is freely available on repositories on the internet. A OSS community has been considered as a virtual (online) community of practice (CoP) [197, 309, 423] which aims to establish a structure where tacit and explicit knowledge is shared and exchanged among various members within a given domain to create a collective value useful to everyone [264, 491]. Developers work on projects that interest them and by so doing, they acquire knowledge associated with their profession. OSS communities offer 24 hours, 7 days a week, 365 days support with up to date content and learning materials, and all of this provided by volunteers at no charge. Therefore, an open source community is more than about software development, but also provides a rich field to explore the process of software knowledge creation, accumulation, and dissemination [423].

Learning in open source communities have been broadly studied in the literature. Hemetsberger and Reinhardt [197, 198] examined how knowledge sharing and learning processes develop at the interface of technology and communal structures of an OSS community. They suggested that knowledge is shared and learned in OSS communities through the establishment of processes and technologies that enable

virtual re-experience for the learners at various levels. They viewed learning in OSS communities as experiential learning whereas learning is a process whereby learning is created through the transformation of experiences as developed by Kolb [247]. Au et al. [25] explored open-source debugging as a form of organizational learning, which heavily relies on adaptive learning [445] to overcome the complexity of software. Singh and Holt [416] provided insights on how the OSS community uses the forums for learning and solving problems. They explored the motivations for joining OSS communities, the learning that occurs in the communities, and the challenges to learning. Hardi [190] had a case study using the Google Chrome project to affirm that situated learning [263] is present among open source developers at an earlier time of a project. Although rapidly growing the current number of studies on learning in OSS communities, the study on the fields of software security is still limited.

9.2.2 The Views on Socio-Technical Aspects

Software systems are not purely technical objects. They are designed, constructed and used by people. Therefore, software systems are components in socio-technical systems, which include technological as well as social structures. The socio-technical aspects provide a deeper analysis of the relationship between the methods, techniques, tools, development environments and organizational structures [20, 21]. There is more and more literature containing applications of the socio-technical systems of software engineering. For example, Lu and Jing [278] present a socio-technical approach to support integrated socio-technical negotiation activities in a collaborative software design process. They address the critical issues of such collaborative negotiation activities, including modeling negotiation arguments based on social and technical factors and analyze these arguments to reconcile the conflicts for software design tasks. Ducheneaut [123] examines the socialization of new members in an open source community using socio-technical analysis since these members interact with both people and material components of a project. Ye et al. [507] propose a socio-technical platform to guide the design of software that supports information seeking and communication during different phases of programming.

Our research is based on the theory of Socio-Technical System (STS) developed by Kowalski [250]. The STS model is depicted in Figure 9.1. This model has two sub-systems include social aspects (culture and structures) and technical aspects (methods and machines). STS model has been applied to evaluate threat modeling in the software supply chain [8], business process re-engineering [4], a framework for securing e-Government services [228] an information security maturity model [21]. The STS provides an appropriate and legitimate way to perform system analysis through a systemic–holistic perspective and helps us understand the intrinsic context in open source phenomenon.

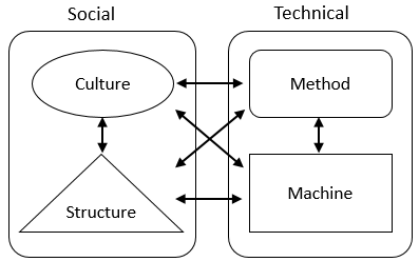


Figure 9.1: Socio-technical system [250]

9.3 Methodology

The research methodology used for this empirical study on OSS communities can be characterized as qualitative research inspired by ethnography. An ethnographic approach typically includes fieldwork done in natural settings, the study of the larger picture to provide a complete context of the activity, an objective perspective with rich descriptions of people, environments, and interactions, and an aim toward understanding activities from the informants' perspective. In empirical software engineering, ethnography provides an in-depth understanding of the socio-technical realities surrounding everyday software development practice [407] and highlights the significance of socio-technical issues in the design of software-intensive systems [36]. The knowledge gained can be used to improve processes, methods, and tools as well as to advance the observed security practices.

9.3.1 Case Selection

To get a broader understanding of the phenomena of interests, we set up the following criteria for the case selection: 1) the selected projects should be community driven; 2) the selected projects should be as diverse as possible; 3) the projects use a wide range of communication tools within the communities. Table 9.1 gives an overview of the selected OSS projects. Having the selected sample cases that cover the range of the diversity of OSS communities is important to refine the phenomena being studied and improve the outcomes of this research endeavor.

Table 9.1: Overview of the selected projects

Project	Age	Software Category	Programming Language
A	3	Collaborative Text Editor	JavaScript
B	8	Content Management System	PHP
C	5	Multimedia playback engine	C/C++

9.4 Data Collection

In terms of data collection, two methods used with qualitative data collection were adopted: observation and interviews.

9.4.1 Observation

The author of this paper participated in the selected projects as an observer to gain a close and intension familiar with the project members and understand the details and processes of the projects. The main idea of this approach is to observe developers performing the activities that they usually do in their daily jobs. To be more specific, observation consists of writing notes about developers' activities, events, interactions, tool usage, and any other phenomena. The digital objects (including source code repository, project documentation, mailing list, code review records, bug reports, and forum) were screened to collect any information related to secure programming. Information collected during the observation was recorded without distracting participants of communities. Observation is an important method to be used in this research because it allowed us to collect information about what learning tools the OSS participants used and how they used them. Moreover, it was a source of valuable insights to assist in a comprehensive understanding of the nature of the case data.

9.4.2 Semi-Structure Interviews

As we wanted to get input from the OSS participants, while still allowing for them to think freely to some extent, we chose to use a semi-structured interview as described by May [290]. Individual interviews were conducted with 13 participants in the selected three projects during the observation period. Participants with short (less than one year), medium (between 1 to 3 years) or long (more than 3 years) experience in the open source development were interviewed. Most of the interviewees did not want to disclose their identity and project name, thus, we did not represent their names in the finding. Due to the geographical distribution of the interviewees, all interviews were carried out via online communication software (Skype and Google Hangout). We had to accommodate all the interviewees' constraints in the setting of interviews. ¨

All interviews were recorded and lasted approximately one hour. The questions were used to understand their experiences in OSS development and examine their perception of learning processes about secure programming in their OSS communities. In order to facilitate elaboration, certain possible follow up questions were prepared beforehand. As we suspected that the subjects would be unwilling to consider themselves behaving insecurely, we also asked about what other members would do. This also has the benefit of covering more subjects.

9.5 Data Analysis

9.5.1 Brief of Case Observations

A. Project A

Project A is an online text editor that allows real-time, collaborative text editing started in mid of 2014. The project website provides basic documentation about what the software does, how to install the software and how to contribute to the project. Security-related information (e.g., coding style, security contact window, etc.) has not yet been published. During the study, it was found that one of the core developers was responsible for security tasks, who solely outlined the security strategy, and had implemented a major portion of security requirements, such as privilege management, user certification, and output control. The source code is routinely refactored in a well-structured format and detailed comments are given in-program segments, which are also embedded critical security designs.

Google Group is used as a discussion forum in the community. No discussion threads about software security were observed until the time we studied. There were four bug reports related to the cross-site scripting vulnerabilities, which were labeled as 'security' or 'XSS' along with developers' discussions and the final code commits. The project held a weekly virtual meeting (Google hangout) to discuss project strategies and roadmaps, and the security issues were brought up during the discussion intermittently Most of the time the attendees were the core developers. The virtual meetings were not recorded, and no documentation was made after meetings. Facebook and Twitter are used to announce new features and release updates.

B. Project B

Project B has been established for 8 years, which aims to provide a web content management system for building robust, flexible websites. Their approach to security is primarily focused on writing quality code, with the objective being to making security a priority. For the purpose, in 2012, they formalized a security team with six official members to coordinate the security tasks of the project. The security team published various documentation to educate the community on security best practices and their development process, including secure coding practices, security risk assessment and peer review to help ensure the products are high quality and secure. In their secure coding practices, for example, they introduce how to sanitize text to avoid improper neutralization input during web page generation (cross-site scripting) and how to use the provided database function to access the database to guard against SQL injection attacks, etc. The security issue is taken seriously in the project. To protect the security of their services, all security issues about the product must be reported directly to a specific email address using PGP encryption. Emails sent to this address are forwarded to the security team's private mailing list. After

evaluating the potential impact and correcting the vulnerability, the security team discloses the security issue with a security advisory.

The project has a set of communication channels to cover different participants and community needs, including mailing lists, IRC chatting, discussion forum, blog, social media (Facebook and Twitter), and yearly face-to-face meetings. To efficient spread security information, a dedicated mailing list is set up for secure communication among users and developers, and a separate channel on IRC for security. The blog is an important medium to share security information with the community, which contains numerous technical documents and papers about the project that are given by the developers, users, and sponsors. The face-to-face meetings are 2 days of events that follow a conference format. The lectures given at conferences were all recorded and provided in conference pages.

C. Project C

Project C is a multimedia playback engine across browsers and media types, which is a community-driven free software effort focused on delivering a high-performance and reliable music player. The project web pages contain answers to frequently asked questions (FAQ) about the software, such as how to build and install software from source and what steam types that the software supports, etc. A wiki website is set up mainly for introducing the development works, including IDE (Integrated Developer Environment) setup, coding guidelines, language bindings, and APIs (Application Programming Interfaces). Since their application may handle data from variable sources that might be possibly untrusted, software security is considered highly critical by the project team. To support security tasks, the project has recruited external security auditors who are responsible for reviewing source code to discover potential security weaknesses, bugs, and violations of programming conventions. Once a bug with potential exploitation is found during the auditing process, the auditing team will coordinate the code owner to handle the bug. The bug fix information will then be posted to the security announcement page, and copies will be sent to the project announce mailing list. The auditing team also acts as a committed reviewer who is committed to the overall quality and correctness of the pull request (submitted code) in GitHub. The most common vulnerability found in the project software is 'Buffer overflow (stack-based and heap-based)', which allows remote attackers to execute arbitrary code via a crafted media file.

Regarding other communication mechanisms within the community, the links of all possible channels are clearly organized in one web page. The mailing list is the main discussion channel within the community. Several mailing lists have been set up for different types of audiences and purposes (e.g. development topics, users topics, announcements, etc.). The project also establishes a question and answer (Q&A) web platform using Stack Overflow, which is mainly for new contributors and software users. For live chatting, they are using Slack for the teams' instant communication tool. There have no questions about software security been asked or discussed in

Q&A or in Slack. The virtual meeting is hosted every two weeks using Bluejeans where the status update of code review is on the routine agenda. Many video-based learning materials were made by the community member and placed on the YouTube project channel.

9.5.2 Arenas for Learning

Opportunities for learning secure programming are not only dependent on the initiative of the learner and the response of other community participants, but also on the arenas where learners meet, communicate and act together. Since OSS communities are all hosted on the internet, their project websites play the central arena that affords learning opportunities.

A. Exploring Project Knowledgebase

In our study, all three communities use a mixed-method to host their project data: the project website and GitHub. GitHub facilitates social coding by providing a web interface to the code repository ("Git") and management tools for collaboration. The project website provides more flexibility in communication within the community, such as Blogs, forums, conference pages, etc.

Documentation provides OSS participants with a shortcut for obtaining an overview of the system or for understanding the code that provides a particular feature. At the very least, it includes instructions on how to get started and details of where to find more information. In our observation, documentation about security knowledge scattered over the community websites. In project B, documentation covers a wide range of security perspectives of the project needs (secure coding practice, risk assessment, etc.) while projects A and C provide only information for software installation and development guidance. One participant from Project B noted that the documentation "had me backing up and restarting several security concepts in web applications, [since] my programming experience is limited to the desktop environment."

Both Project B and C provided video-based learning materials for participants (recorded from a conference or homemade video) on the project website or YouTube respectively. Watching video is viewed stand-alone from other forms of training requiring no interaction from the learners. With the explosion of video-sharing services such as YouTube, the amount of recorded audiovisual information has grown exponentially in open source communities. Respondents gave opinions about learning software security from watching the videos:

"It [Video] allows me to attend the lecture on a flexible schedule and move at my own pace."

It is noteworthy that some technical talks with security topics in the conference are recorded and provided on YouTube, however, few respondents claimed that they reached the video and learned about secure programming from it.

"…It costs me much time to watch the conference video full of contents"

"It is good there is transcription. I can search [keyword] on it."

Reading release notices (security advisories) can be an opportunity for exploring security knowledge. A security advisory is a way for open source communities to communicate security information to the public. Usually, it involves updating to a new release of the code that fixes the security problem. From reading the security advisory, learners can learn not only security enhancements or changes that are related to security vulnerabilities but guidance and mitigations that may be applicable for publicly disclosed vulnerabilities. In Project B, the security-related issues are kept secret until the advisory is ready to be released, at which point it is publicized widely so that all developers and users can address it quickly. A respondent indicated:

"Because it was short, to the point, and very accountable"

B. Reading Source Code

The source code is seen as the actual documentation while the other kinds of documentation are informally produced to support situated discussion. Reading source code is a key activity in OSS maintenance. Developers can profitably apply experiences and reading systems from text databases. Most interview respondents agree that studying the source code of the project expands the horizons of software security. The respondents stated that:

"For me, reading source code is about learning new strategies for solving security problems."

"If you check the PHP code of the Symfony framework, for example, you will know about dependency injection, events, the model/view/controller pattern, and so on."

It is found that only code authors have intentions to highlight what they have done about security in their code via medialization or enough commenting, code reader hardly learn software security from it. Wide gulf in knowledge and experience between security experts and beginners can be a barrier to learn software security efficiently from source code. Some respondents expressed the difficulties in learning security from source code. Some comments are collected:

"It would take 1 to 2 years [for new contributors] to capture the coding patterns and algorisms…to understand the meanings behind the code."

"When I was a junior developer I used to lament the lack of comments in code created by team members. Now that I am a lot more experienced, many comments tend to clutter up the code and reduce the comprehensibility."

"If you're a first-time web developer and check on GitHub repository saying something like "CSRF vulnerability", it can often be hard to tell what exactly that is, what impact it has on your application, and how to fix it.."

From project management perspectives, since the new code is merged into the main codebase irregularly, to keep the source code in a good level of readability is a challenge to OSS maintenance. One respondent of Project A commented:

"We have to spend extra time doing codebase refactoring and commenting, to make the code cleaner, simpler and readable."

C. Following mailing list and forum

Mailing lists and forums are common places for community communications to discuss requirements of the software or development issues, and they are also the main places to provide support to users. These asynchronous communication technologies are not only valuable for knowledge creation purposes, but also in order to make community members think before they act and respond. The respondents commented about the mailing list and forum (Project B & C):

"Many of my problems are solved by just browsing through other people's questions."

"…more than questions and answers. The messages contain the path leading to answers to the question."

"Though if you look at mailing lists the project has an extensive discussion about the security architecture whenever somebody has a problem or wishes to change something."

Some respondents expressed they felt a bit frustrated with the gap between the community expertise and their own. They either hesitate to ask questions or cannot catch up the pace of discussions.

"Once you have enough knowledge, it can help."

Furthermore, threaded discussions could be a barrier to secure programming learning, especially for the mailing list, as indicated by respondents:

"If you subscribe to a very active list, you could easily pile up several hundred emails in a day."

"A lot of irrelevant information to sift through."

In Projects B, although a mailing list is set up specifically for 'security', it is mainly for vulnerability reporting. Such a mailing list cannot be subscribed by participants, and can only be accessed by the dedicated members.

D. Engaging in code review

Although in OSS development, a programmer may write a complete program independently from other programmers, the software component that will be still examined by other software engineers. Pull requests (PRs) and coding review represents a form of learning processes in which knowledge is created collectively in a distributed work process. It might shortly address the role of PRs as being a workflow mechanism for a developer to notify team members that a feature or fix, developed on a separate branch, is ready. This lets everybody involved know that he (or she) can review the code, providing a forum discussing the implementation of the proposed feature. Questions, answers, and discussions about the issues are communicated back and forth between the community and the members. The respondents had the following comments about code review (pull requests):

"I learned [about security] in this project after constantly getting feedbacks on my pull requests regarding how this kind of code can go wrong."

"It [code review] is an effective learning process for me – a chance to see what mistakes I made and the bad habits that I had usually."

9.6 Discussion

We summarize and classify the research findings based on the STS model presented in section 9.2.2. Figure 9.2 gives an overview representation of the socio-technical model in this study. In the following sections, we give a detailed discussion of each part.

Culture	**Method**
• *Security culture / value*	• *Self-directed learning* • *Learning from mistakes*
Structure	**Machine**
• *Security expertise coordination*	• *Non-unified security knowledge sharing mechanisms*

Figure 9.2: A socio-technical analysis of findings

9.6.2 Self-Directed Learning

Our study found that learning processes of secure programming for OSS developers are centered on reactive and self-directed learning experiences. OSS learners may want to learn according to what provokes their curiosity instead, and that may mean

starting from the middle and proceeding to pick out material in order of interests or the level of competency instead of what is being defined by a typical structure. The learning journeys they experience are usually unstructured and non-linear. The openness and transparency of OSS projects provide an interesting setting for participants to exercise self-directed learning. In OSS development, participants usually first try to solve their problems themselves by the mean of available materials and if required by exploring the web: browsing documentation (guideline, wiki, FAQ, etc.), studying the source code and engaging in discussion threads. These internet resources have the advantage to provide the community with an information infrastructure for publishing and sharing a description of software development in the form of hypertext, video, and software artifact content indexes or directories.

9.6.3 Learning from Mistakes

OSS developers care more about making the software work eventually rather than trying to make it work the very first time. When contributing to the projects, they mostly focus on their immediate goals that usually involve functional requirements and system performances. Daniela et al. [336] point out that software vulnerabilities are blind spots in developers' heuristics in their daily coding activities. They have not considered the importance of a given function might have to the overall security of their application until they made mistakes and understand the consequence of the flaw. The learning ability from mistakes becomes essential in this context.

The processes of pull requests and code review, for example, are important enablers for developers to reflect their code, take corrective actions and build concrete experiences, most importantly, learn from the mistakes. As noted by one respondent:

"We cannot write code properly, so we need someone to pair with us to smooth our failures."

Subsequently, not only members are opened about their mistakes, they share their experience as learning opportunities for others. This is also helpful for those who have not yet suffered through the same mistakes on the road. Researchers have also indicated that engagement with mistakes fosters the secondary benefits of deep discussion of thought processes and exploratory active learning [404]. When the correct answer is made to the mistakes, though, and people appreciate that the answer is correct as well as why that answer is correct, they are able to integrate that information into memory and improve performance [19].

9.6.4 Non-Unified Security Knowledge Sharing Mechanisms

A major problem we found is a lack of sufficient as well as efficient knowledge sharing mechanisms for secure programming in OSS communities. Like our study, the security knowledge is scattered over the community websites (source code, documentation, wiki, forum, conference pages, etc.), and the quantity of transferred knowledge is varied by projects. Finding and learning knowledge about secure

programming becomes a key challenge that is highly dependent on the resources the community provides. For example, to keep all the code in the repository is in a consistent style, OSS projects normally publish their own coding guideline, a set of conventions (sometimes arbitrary) about how to write code for that project. However, they rarely to address the security requirements in documentation to help drive the team to understand the prioritized security needs of the entire project. Some security talks embedded in the recorded video (conference or learning materials) without proper indication (resource location, topic indexing) also creates a barrier for participants to learn about secure programming. Newcomers feel that comprehending systems from exploring the website is hopeless, so they as well prefer to start with programming. They lose the learning opportunity about the security requirements of the project and being aware of the possible mistakes they may make in their code.

9.6.5 The Need for Security Expertise Coordination

People join the OSS community at a different age, with different backgrounds, different capacities and resources, and with different objectives. The issue of security expertise levels among OSS development members is critical, especially when considering a variety of domains of expertise ranging from strategic goal and problem-solving expertise to trained motor skills and operational expertise. Another problem is that secure programming is still not a well-known discipline in OSS communities, so there remains a lot of confusion as to what is secured code and what the project wants.

In our study, we observe that effective learning firstly results in coordinating necessary security expertise in the project, which enables a high level of security knowledge creation and the satisfaction of the learning process. In this study, expertise coordination is manifested through the two following strategies: coordinating organizational structure and security infostructure. Every member involves in OSS development should be concerned with software security, but it is inefficient to demand each participant taking care of all security aspects. The coordinating organizational structure serves as subject matter experts to ensure that security-related issues receive necessary attention in the community. OSS communities can utilize security experts to define security requirements and best practices, help perform code reviews, and provides the necessary education for the software development staff [209]. Through this structural mechanism, security knowledge is able to gain valuable insights from the organization to facilitate strategic decision making [229].

The term infostructure is commonly used to describe the infrastructure of information that is used in multiple disciplines. As indicated by Tilton [439], and infostructure is the layout of information in a manner such that it can be navigated – it is what's created any time an amount of information is organized in a useful fashion. In the knowledge sharing process, infostructure serves as a role to provide rules, which

govern the exchange between the actors on the network providing a set of cognitive resources (metaphors, common language) whereby people make sense of events on the network [345].

The role of expertise coordination is to provide access to the experience and knowledge, which help the learners reach their potentials. Software security knowledge can be abstracted, explicitly represented, codified, and accessed. In OSS development, security expertise coordination not only facilitates a common understanding of security requirements but also specializes in the context of the project development. It helps participants identify the location of the security information, and the most important for knowledge work, knowing where an answer to a problem.

9.6.6 Security Culture is the Key

Security culture reflects the belief and values of people that make up the organization. It is about actively practicing good security habits and making security-minded decisions [105, 288]. In short, security culture is the way our minds are programmed that will create different patterns of thinking, feeling, and actions for providing the security process [7]. It also includes all socio-cultural measures that support technical security methods in order for making security a natural aspect of organizational members' daily activities [397].

Culture shapes what a group defines as relevant knowledge, and this will directly affect which knowledge a unit focuses on [112]. Similarly, security culture decides how much security knowledge is disseminated within the community and what knowledge learners can learn. If an OSS project truly holds a value that software security is important, then particular behaviors and actions can be expected. Like our study, the three projects, with different software domains and project stages, unfolding different security culture: Project A is at the young age and rapid production of function-based requirements was their strategy; Project B helps educate the community on writing secure code, and Project C focuses on proactive security auditing. The security culture backgrounds either at organizational or at the individual level have impacts on the amount of security knowledge transferred within the community, further, affecting participants' learning processes.

9.7 Limitation

The study has some limitations. The observation was conducted in three OSS projects. It is reasonable to think that observation in several projects, covering more software types, could have given a more balanced result in the form of highlighting both hindrances and support for secure programming learning. Data collection and a major part of the analysis were conducted by the first author. More participation by different observers could have broadened the view of observations in the OSS communities.

9.8 Conclusion

This empirical study focuses on exploring learning about secure programming in open source software communities. Open source software has become a critical component and a key competency of the information and communication technology (ICT) ecosystems. While the number of found vulnerabilities in OSS is increasing, it is noteworthy that knowledge sharing and learning about secure programming in OSS communities have not gained much attention, and it is necessary to examine why knowledge-sharing mechanisms have not been effective despite efforts by OSS projects.

This study first addressed the learning opportunities for secure programming in OSS communities while investigating how mechanisms for sharing security knowledge have been implemented. Specifically, this study applied a socio-technical systems perspective, which systematically and holistically took into account the social context as well as technological aspects. Based on the socio-technical framework and context, we then examined the main factors that were once disproportionately considered in the learning process of secure programming in the context of OSS development and made suggestions for promoting a more effective as a central role for building robust software products.

In the context of distributed development, like OSS projects, learning is always to a great deal an individual exercise. People join the OSS community at a different age, with different backgrounds, different capacities and resources, and with different objectives. The fields they came from are from any discipline that might lack formal, college-level software security training, they do not see an economic incentive for squeezing security thinking into their works and producing secure code. It is suggested that OSS communities have to establish rules and norms, roles and facilities, i.e., to offer opportunities for learning and self-development of secure programming knowledge for newcomers as well on the horizontal level between the experienced (but ever-learning) community members.

9.9 Acknowledgment

The author would like to thank Professor Dr. Stewart Kowalski and Professor Dr. Rune Hjelsvold of Faculty of Information Technology and Electrical Engineering at Norwegian University of Science and Technology, who have made comments and suggestions in this paper.

Chapter 10

An Empirical Study on Security Knowledge Sharing and Learning in Open Source Software Communities

Wen, Shao-Fang. "An Empirical Study on Security Knowledge Sharing and Learning in Open Source Software Communities." *Computers*, 2018, volume 7, issue 4.

Abstract— Open source software (OSS) security has been the focus of the security community and practitioners over the past decades. However, the number of new vulnerabilities keeps increasing in today's OSS systems. With today's increasingly important and complex OSS, lacking software security knowledge to handle security vulnerabilities in OSS development will result in more breaches that are serious in the future. Learning software security is a difficult and challenging task since the domain is quite a context-specific and the real project situation is necessary to apply the security concepts within the specific system. Many OSS proponents believe that the OSS community offers significant learning opportunities from its best practices. However, studies that specifically explore security knowledge sharing and learning in OSS communities are scarce. This research is intended to fill this gap by empirically investigating factors that affect knowledge sharing and learning about software security and the relationship among them. A conceptual model is proposed that helps to conceptualize the linkage between socio-technical practices and software security learning processes in OSS communities. A questionnaire and statistical analytical techniques were employed to test hypothesized relationships in the model to gain a better understanding of this research topic.

10.1 Introduction

Open source software (OSS) is based on the principle that computer programs should be shared freely among users, giving them the possibility of introducing improvements and modifications. OSS is at the core of today's information technology (IT) infrastructure and information systems; about 80% of companies run their operations on OSS [330] and 96% of applications utilize OSS as the software components [50]. OSS is developed collectively by an online community of practices (CoPs) with a strong relationship between the social and technical interactions in a decentralized and knowledge-intensive process [198, 245]. Groups of volunteers participate in communities that are essential for OSS project development. They collaborate and integrate expertise to solve particular programming problems, as well as to deliver and maintain the software that is produced by the OSS community [138, 140, 391].

OSS security has been the focus of the security community and practitioners over recent decades. Many studies have been conducted by both researchers and practitioners on the mechanisms of building security in OSS development [481]. However, the number of new vulnerabilities keeps increasing in today's OSS systems. The Blackduck 2017 Open Source Security and Risk Analysis report announced that 3623 new OSS vulnerabilities occurred in 2016—almost 10 per day on average and a 10% increase from 2015 [50]. These vulnerabilities open some of the most critical OSS projects to potential exploitation such as Heartbleed and Logjam (in OpenSSL); Quadrooter (in Android); Glibc Vulnerability (in Linux servers and web frameworks); NetUSB (in Linux kernel), and many others [272, 357]. With today's increasingly important and complex OSS, lacking software security knowledge to handle security vulnerabilities in OSS development will result in breaches that are more serious in the future.

Building secure applications is a complex and demanding task that developers often face. Knowledge of software security is more than simply having a checklist or reminders of things [31]. It is about understanding the potential security risks that are induced by the software, and how to manage them [294]. Comparing with proprietary software development in enterprises, which usually involves formal training and practices about secure software development, OSS development relies mainly on the ability of participants themselves to acquire, refine, and use new aspects of security knowledge to fulfill the needs of their work in the community. Much of an OSS community's security knowledge lies within its documents, discussions, decisions, processes, and the awareness by members of other members' expertise. Both finding and learning the security requirements and practices of the project become key challenges that are highly dependent on the knowledge resources the community provides. Many OSS proponents believe that the OSS community offers significant learning opportunities from its best-practices [204, 257], which are different from the education of traditional models [71, 144]. However, studies that specifically explore security knowledge sharing and learning in OSS communities are scarce.

As there is still a dearth of empirical research into security knowledge learning in the context of OSS development, this study intends to fill this gap by empirically investigating factors that affect knowledge sharing and learning about software security and the relationships among them. The purpose is twofold. Firstly, we are interested in obtaining a deeper understanding of how factors complement each other in shaping security knowledge sharing and learning behavior. Secondly, we suggest a conceptual framework that includes both social (security culture) and technical (security expertise coordination) constructs to investigate how OSS communities can shape this behavior. We attempt to fulfill this purpose by utilizing a questionnaire survey and statistical analytical techniques on OSS project participants. The data analysis result is the main contribution of the paper. This is presented as a preliminary research model, which includes a set of socio-technical constructs that could potentially describe security knowledge sharing mechanisms and learning processes in OSS communities.

The rest of this paper is structured as follows. Section 10.2 describes the theoretical background of the research. The conceptual framework defining the constructs and hypothesized relationships are depicted in Section 10.3. The research method is explained in Section 10.4. In Section 10.5, we present the result of data analysis. Section 10.6 provides a discussion based on the result. We describe the conclusion and limitation of this study in Sections 10.7 and 10.8 respectively.

10.2 Theoretical Background

10.2.1 Knowledge Sharing

Christensen [207] defines knowledge sharing as a process that exploits existing knowledge by identifying, transferring, and applying it to solve tasks better, faster, and cheaper. It is 'the process of transferring knowledge from a person to another in an organization' [347]. Knowledge sharing is a deliberate act that makes knowledge reusable by other people through knowledge transfer [267]. It is about "how people share and use what they know" [249] and requires the active engagement of individuals in a process of interaction and learning [374]. As Nonaka [327] points out, knowledge is created and expanded through social interaction between people and their creative activities [327]. Through knowledge sharing, individuals could exchange tacit or explicit knowledge, hence, together create new knowledge [450].

Terminologies such as 'knowledge distribution' and 'knowledge transfer' are also used for referring to knowledge sharing and bring paronomasia; e.g., Haas and Hansen [186], Christensen [207], Cabrera et al. [66], Wasko and Faraj [475], and Inkpen and Tsang [215]. Although these definitions and discussions of knowledge sharing vary from different perspectives, they do deliver a similar core concept, which is using existing knowledge within the organization to solve problems, generating new learning, and empowering the organization for innovation.

10.2.2 Knowledge Sharing and Learning in OSS Communities

The purpose of the OSS community is essentially knowledge sharing and collaboration [268]. An OSS community has been considered as a social leaning CoP [197, 309, 423], which aims to establish a structure where tacit and explicit knowledge is shared and exchanged among various members within a given domain to create a collective value useful to everyone [264, 491]. Developers build the software by relying on extensive peer production and through the skillful use of the software and communication tools available on the Internet [260]. They share and acquire knowledge associated with their profession. Furthermore, OSS communities have been a source of learning for participants since their creation [416], which offer 24 h, 7 days a week, 365 days support with up to date content and learning materials, and all of this provided by volunteers at no charge. Therefore, an open source community is more than about software development, but also provides a rich field to explore the process of software knowledge creation, accumulation, and dissemination [423].

Knowledge sharing and learning in open source communities have been broadly studied in the literature. Sowe et al. have introduced a knowledge-sharing model to develop an understanding of the dynamics of collaboration and how knowledge sharing is distributed over OSS development teams [422, 424]. Chen, Xiaogang et al. adopted the perspective of the transactive memory system (TMS) to empirically examine the possible team cognitive mechanisms that facilitate knowledge sharing in OSS communities [75]. Their study showed that communication quality positively influences the knowledge sharing and technical performance of the team. Iskoujina and Roberts investigated the factors that motivate participants to share their knowledge in OSS communities and concluded that the quality of management influences the extent to which the motivations of members actually result in knowledge sharing [217]. Chen, Xiaohong analyzed key factors affecting knowledge sharing in OSS projects, which include participative motivation, social network, and organizational culture [76, 77].

Au et al. explored open-source debugging as a form of organizational learning [25], which heavily relies on adaptive learning [445] to overcome the complexity of software. Singh and Holt provided insights on how the OSS community uses the forums for learning and solving problems. They explored the motivations for joining OSS communities [416], the learning that occurs in the communities, and the challenges to learning. Hardi had a case study using the Google Chrome project [190] to affirm that situated learning [263] is present among open source developers at an earlier time of a project. Hemetsberger and Reinhardt examined how knowledge sharing and learning processes develop at the interface of technology and communal structures of an OSS community [197, 198]. They suggested that knowledge is shared and learned in OSS communities through the establishment of processes and technologies that enable virtual re-experience for the learners at various levels. They viewed learning in OSS communities as experiential learning whereas learning is a

process whereby learning is created through the transformation of experiences as developed by Kolb [247].

10.3 Conceptual Framework

The conceptual framework is developed based on the author's prior ethnographic study on three OSS communities [483]. The study applied a socio-technical systems perspective [250], which systematically and holistically took into account the social context as well as technological aspects. The observation result was analyzed and categorized with social (culture and organization structure) and technical (method and machine) aspects. Figure 10.1 depicts the conceptual and theoretical structure that includes four constructs, namely: security culture, expertise coordination, security knowledge sharing, and software security learning. The background of the conceptual framework is described below.

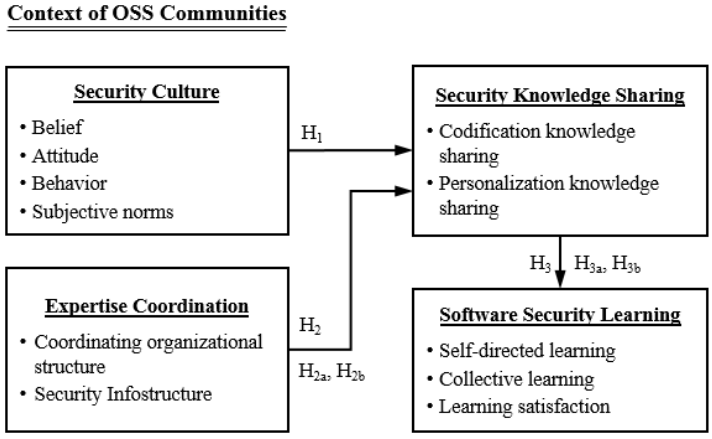


Figure 10.1: The conceptual framework.

10.3.2 Security Culture and Security Knowledge Sharing

Security culture is recognized in the security community and scientific literature as one of the most important foundations of organizational security. In short, security culture is the way our minds are programmed to create different patterns of thinking, feeling, and actions for providing the security process [7]. Security culture is based on the interaction of people with information assets and the security behavior they exhibit within the context of the organizational culture in the organization [105]. Security culture involves identifying security-related ideas, beliefs, and values of the group, which shape and guide security-related behaviors [364]. Martins and Eloff define information security culture as the perceptions, attitudes, and assumptions that are accepted and encouraged by employees in an organization in relation to information security [288]. Ngo et al. suggest that security culture is the accepted

behavior and actions of employees and the organization as a whole, as well as how things are done in relation to information security [325]. Therefore, the four main aspects of security culture formed in this study are:

Belief: An acceptance or a firmly held opinion that security is of value to the community.

Attitude: A feeling or emotion toward various activities that pertain to the security of the software product produced by the community.

Behavior: Actual or intended activities and risk-taking actions in secure software developments.

Subjective norms: A combination of perceived expectations from relevant individuals or groups along with intentions to comply with security-related expectations.

Organizational culture has been shown to influence the success of knowledge management practices [112, 124, 132, 150]. Culture shapes what a group defines as relevant knowledge, and this directly affects the knowledge a unit focuses on [112]. In the context of information security, security culture decides how much security knowledge is disseminated within the community and what knowledge learners can learn. The security culture backgrounds either at organizational or individual levels impact on the amount of security knowledge transferred within the community, further affecting the participants' learning processes. Thus, the research hypothesis is as follows:

Hypothesis H_1. *Security culture is positively associated with security knowledge sharing.*

10.3.3 Expertise Coordination and Security Knowledge Sharing

Expertise coordination is the process of knowledge integration and the outcome of exchanging and combining knowledge through interactions among team members [371]. Expertise coordination is believed to serve as an important component of software development. According to the findings of empirical studies in the literature, expertise coordination strongly influences project performance, team effectiveness, and team efficiency in software development projects [134, 222, 252, 440]. This has a bearing on both the physical and virtual development teams [99, 133, 200]. For complex non-routine intellectual tasks, expertise coordination (the management of knowledge and skill of dependencies) is necessary so that the software team can recognize where expertise is located, needed, and accessed [134]. A great challenge of security expertise coordination is to combine explicit and tacit knowledge in all management and security expert decisions, and to get knowledge moved from individuals within the whole organization between different actors, and from tacit domain to explicit domain and also vice versa [21]. In this study, expertise coordination is manifested through the two following strategies: coordinating organizational structure and security infostructure.

A. Coordinating Organizational Structure

The organizational structure supports the assignment of both technical and human resources to the tasks that must be done and provide mechanisms for their coordination [81]. It also establishes and enables strategic- and operational decision-making, monitoring of performance, and operating mechanisms that transfer directives on what is expected of organizational members and how the directives should be followed [81]. The organizational challenges faced by OSS projects are significant because the project must deal not only with problems faced by any software development process but also with the complexity of coordinating efforts of a geographically distributed base of volunteers working on the software [322]. OSS projects usually utilize security experts to define security requirements and best practices, help perform code reviews, and provides the necessary education for the software development staff [209]. The coordinating organizational structure serves as a subject matter expert to ensure that security-related issues receive necessary attention in the community. Through this structural mechanism, security knowledge is able to gain valuable insights from the organization to facilitate strategic decision making [229].

B. Security Infostructure

The term infostructure is commonly used to describe the infrastructure of information that is used in multiple disciplines. As indicated by Tilton, an infostructure is the layout of information in a manner such that it can be navigated—it is what is created any time an amount of information is organized in a useful fashion [439]. In the knowledge sharing process, infostructure serves as a role to provide rules, which govern the exchange between the actors on the network providing a set of cognitive resources (metaphors, common language) whereby people make sense of events on the network [345]. In the context of OSS development, developers contribute from around the world, meet face-to-face infrequently if at all, and coordinate their activity primarily by means of digital channels on the internet [98, 368]. A proper infostructure can help learners identify the location of the security information, knowing where an answer to a problem is located, and acquiring as much knowledge as possible [358].

Based on the above discussion, the research hypotheses are given as follows:

Hypothesis H_2. *Expertise coordination is positively associated with security knowledge sharing.*

H_{2a}: *Coordinating organizational structure has a positive effect on security knowledge sharing.*

H_{2b}: *Infostructure has a positive effect on security knowledge sharing.*

10.3.4 Security Knowledge Sharing and Software Security Learning

Learning may be the most strategically valuable dynamic capability [437]. Learning is the process by which knowledge comes into being and is enhanced over time, and is therefore intimately associated with knowledge sharing process [297]. Learning experts argue that online knowledge sharing can be regarded as an important form of collective learning [377]. In OSS projects, the fundamental functionality for security knowledge sharing is to capture security experts' knowledge in the project repository that other project participants can access and learn about software security. Knowledge sharing can be facilitated by the project-based organization by using codification or personalization mechanisms [51].

A. Codification Security Knowledge Sharing

The codification knowledge sharing mechanism captures individual or group-held knowledge and makes it the wider property of the organization [51], which facilitates a setting for participants to exercise self-directed learning. The basic functionality for this knowledge sharing mechanism is to capture security experts' knowledge in the project repository that other project participants can access and learn about software security, which provides a setting for participants to exercise self-directed learning. Moreover, the internet resources have the advantage to provide the community with an information infrastructure for sharing codification materials of software development in the form of hypertext, video, and software artifact content indexes or directories. These codification materials (documentation, wiki, release notices, security advisories, source code, etc.) provides the participants with a shortcut for obtaining an overview of the system or for understanding the code that provides a particular feature. At the very least, it includes instructions on how to get started and details of where to find more information.

B. Personalization Security Knowledge Sharing

Personalization knowledge sharing provides communications in another form, as it is concerned with the use of people as a mechanism for sharing knowledge [24]. Personalization as a knowledge-sharing mechanism has the inherent flexibility of transmitting tacit knowledge, and allowing for discussions and sharing interpretations that may lead to the development of new knowledge [51]. OSS communities adopt various forms of technologies, such as mailing lists, forums, and Internet Relay Chat (IRC) to support knowledge sharing via personalization mechanisms. These technologies provide useful means of storage and acquisition for the communities' experiential knowledge, given that individuals have a general preference for obtaining information from other people, rather than from documents [334]. Although in OSS development, a programmer may write a complete program independently from other programmers, the software code will be still examined by other software engineers. The coding review also represents a form of personalization knowledge sharing mechanism in which knowledge is created collectively in a

distributed work process. The peer-review process emphasizes the importance of collecting learning and shared dialogue [423]. During code review, questions, answers, and discussion about the coding issues are communicated back and forth between the community and the members. Developers have the opportunity to reflect their code, take corrective actions and build concrete experiences in the code review process.

Based on the above discussion, the following hypotheses were made:

Hypothesis H_3. *Security knowledge sharing is positively associated with software security learning.*

H_{3a}: *Codification knowledge sharing has a positive effect on software security learning.*

H_{3b}: *Personalization knowledge sharing has a positive effect on software security learning.*

10.4 Methodology

This research adopted a quantitative approach to a survey research method to investigate the relationships among security culture, expertise coordination, security knowledge sharing, and software security learning. A self-administered Web-based survey was used to collect individual-level perception data from participants in OSS projects. The use of an OSS participant survey was deemed appropriate to test the hypotheses outlined in the previous section.

10.4.1 Instruments

The survey instrument used in this study was the outcome of an iterative process of checking and refinement. The constructs and items used to operationalize the research were developed following the generally accepted guidelines of reliability and validity or multiple-item measures [332]. After synthesizing the results of the literature review, a questionnaire was developed based on the structure of the research framework. Some survey questions were inspired by existing studies, while others were created specifically to suit the research context of our study. For the measurement instrument of key variables, each item was measured on a five-point Likert scale (Cf. Appendix). The primary references for the constructs and items used in this study are summarized in Table 10.1.

10.4.2 Data Collection

Samples for the empirical study were randomly collected from participants in OSS development projects, available on GitHub. GitHub is an online database of open source software projects. Users and potential contributors can access information about the projects and download current versions of the software being developed.

Table 10.1: Measurement instrument for key variables in the questionnaire.

Construct		Item	Reference
Security culture	Belief	Security value, cognition	[105]
	Attitude	Risk-taking, responsibility	
	Behavior	Secure coding, compliance	
	Subjective Norms	Peer influence, expectation	
Expertise coordination	Coordinating structure	Security expert, assistance	[67]
	Security Infostructure	security website, navigation, taxonomy	
Security knowledge sharing	Codification knowledge sharing	Documentation, multimedia,	[5]
	Personalization knowledge sharing	Experience, collaboration	
Software security learning	Self-directed learning	Exploration, search	[158]
	Collective learning	Feedback, problem-solving	
	Learning satisfaction	Enjoyment, simplicity	

As of April 2017, GitHub reports having almost 20 million users and 57 million repositories [163], making it the largest host of source code in the world [172]. The anonymous questionnaires were sent via e-mail to a list of OSS participants at the beginning of August 2017. The data collection period lasted 3 months and 402 questionnaires were completed. Among them, 324 were valid; and another 78 respondents were discarded due to the reason that they did not participate in any open source community. Table 10.2 shows demographic information about the sample, which includes gender, age, and seniority in the community and product categories of the projects.

10.4.3 Reliability and Validity Analysis

Validating constructs is important before any further analysis is conducted. To this end, reliability and validity tests were carried out following the sequence and approach that was taken by Straub [429]. Table 10.3 outlines the results of the reliability and validity tests performed on the survey items. Convergent validity, the degree to which multiple attempts to measure the same concept are in agreement, was evaluated by examining the factor loading within each construct, composite reliability, and variance extracted [4, 187]. We used confirmatory factor analysis (CFA) with AMOS to examine the convergent validity of each construct. The factor loadings range from 0.493 to 0.872, and these are greater than the recommended level of 0.35, which is based on 250 samples and a 0.05 significance level [187]. All composite reliabilities and variance-extracted measures of constructs exceed the recommended level of 0.8 and 0.5 each. The reliability of a scale (factor or construct) is to examine its internal consistency by calculating Cronbach's alpha. This method indicates the extent to which items within a scale are homogenous or correlated [27].

Table 10.2: Demographic characteristics of the respondents (n = 324).

Item	Category	Frequency	Percentage
Gender	Male	289	89.2%
	Female	23	7.1%
	Prefer not to say	12	3.7%
Age	<20	13	4.0%
	20–30	147	45.4%
	31–40	116	35.8%
	41–50	35	10.8%
	>50	7	2.2%
	Prefer not to say	6	1.9%
Seniority in the community	<3 months	13	4.0%
	3–6 months	17	5.2%
	7 months–1 year	47	14.5%
	2–3 years	89	27.5%
	>3 years	158	48.8%
Product Category	Healthcare, Health Tech	12	3.7%
	Science, Geospatial, Astronomy	9	2.8%
	Retail & E-Commerce	7	2.2%
	Big Data, AI, BI, Machine Learning	22	6.8%
	Enterprise Software	11	3.4%
	Mobile Apps	19	5.9%
	Gaming, Entertainment, Media	13	4.0%
	Financial Services	15	4.6%
	Development Framework	35	10.8%
	Internet, email, browser, content management	43	13.3%
	Database, file system	30	9.3%
	Security, firewall, anti-virus, encryption	27	8.3%
	Operating system	21	6.5%
	Education, knowledge management, eLearning	19	5.9%
	Internet of things	28	8.6%
	Others	13	4.0%

It is also reflective of the consistency between different items on a scale, in measuring the same attribute. The resulting alpha values ranged from 0.827 to 0.907, which were above the acceptable threshold (0.70) suggested by Nunnally [332]. From the analyses mentioned above, it was found that the survey items on each construct met the requirements for reliability and validity.

Table 10.3: The convergent validity and reliability test results.

Construct		Item	Convergent Validity (Factor Loading [1])	Reliability (Cronbach's α)
Security culture	Belief	Security value	0.727	0.873
		Cognition	0.651	
	Attitude	Risk-taking	0.736	
		Responsibility	0.814	
	Behavior	Secure coding	0.801	
		Compliance	0.735	
	Subjective norms	Peer influence	0.781	
		Expectation	0.665	
Expertise coordination	Coordinating structure	Security expert	0.674	0.827
		Assistance	0.523	
	Security infostructure	Security website	0.818	
		Navigation	0.798	
Security knowledge sharing	Codification knowledge sharing	Documentation	0.746	0.907
		Multimedia	0.812	
	Personalization knowledge sharing	Experience	0.728	
		Collaboration	0.727	
Software security learning	Self-directed learning	Exploration	0.831	0.883
		Search	0.736	
	Collective learning	Feedback	0.753	
		Problem-solving	0.851	
	Learning satisfaction	Enjoyment	0.493	
		Simplicity	0.627	

[1] Factor loadings are from confirmatory factor analysis.

10.5 Analysis and Result

Statistic software SPSS 24.0 for Windows was used to analyze the data. Pearson's correlation analysis and multiple regression analysis were to analyze security culture, expertise coordination, security knowledge sharing, and software security learning.

10.5.1 Relationship between Security Culture and Security Knowledge Sharing

This study adopted Pearson's correlation analysis to determine the correlation between security culture and security knowledge sharing. Table 10.4 shows that the correlation coefficient between security culture and security knowledge sharing is 0.671, a highly positive correlation. The correlation coefficients of each of the security culture factors—belief, attitude, behavior, and subjective norms are 0.591, 0.628, 0.427, and 0.584 respectively. Regarding the correlation among all security culture factors, the results show a strong correlation among them that reaches a significant level ($p <$

Table 10.4: The correlation analysis for security culture and security knowledge sharing.

		Security Knowledge Sharing
Security culture	Pearson correlation	0.671 **
	Sig. (2-tailed)	0.000
Belief	Pearson correlation	0.591 **
	Sig. (2-tailed)	0.000
Attitude	Pearson correlation	0.628 **
	Sig. (2-tailed)	0.000
Behavior	Pearson correlation	0.427 **
	Sig. (2-tailed)	0.000
Subjective norms	Pearson correlation	0.584 **
	Sig. (2-tailed)	0.000

** Correlation is significant at the 0.01 level (2-tailed).

0.01). Thus, security culture has a significant positive correlation with security knowledge sharing. Hence, hypothesis H_1 is proven.

10.5.2 Relationship between expertise coordination and security knowledge sharing

Table 10.5 indicates that expertise coordination has a significant positive correlation with security knowledge sharing in which Pearson correlation is 0.400 and $p < 0.01$. The correlation coefficients of expertise coordination factors-coordinating organizational structure and infostructure are 0.628 and 0.584 respectively. The results showed a strong correlation among all expertise coordination factors that reached a significant level ($p < 0.01$). Consequently, the research result favored hypothesis H_2, the stronger coordinating organizational structure and security infostructure, the higher the security knowledge sharing degree. Hence, H_{2a} and H_{2b} are also proven valid.

Since expertise coordination has a significant correlation with security knowledge sharing, this study used multiple-regression analysis to understand the linear relationship between a group of forecast variables and a valid variable. The multiple-regression analysis used in this research is shown in Table 10.6. As indicated in the table, B value, Beta, and *t*-value have positive values. The prediction equation is based on the unstandardized coefficients, as follows: $y_1 = 2.418 + 0.151x_3 + 0.217x_4$ (where x_3 is coordinating organizational structure and x_4 is security infostructure). All variables show a positive relationship. Looking at the *p*-value for each variable, the predictor variables of coordinating organizational structure and security infostructure not statistically significant because of both of their *p*-value greater than 0.05. In this model, the two factors do not provide a significant impact on security knowledge sharing. Thus, given the above relationship, hypotheses H_{2a} and H_{2b} are partially supported.

Table 10.5: The correlation analysis for expertise coordination and security knowledge sharing.

		Security Knowledge Sharing
Expertise coordination	Pearson correlation	0.400 **
	Sig. (2-tailed)	0.000
Coordinating organizational structure	Pearson correlation	0.376 **
	Sig. (2-tailed)	0.000
Security infostructure	Pearson correlation	0.370 **
	Sig. (2-tailed)	0.000

** Correlation is significant at the 0.01 level (2-tailed).

Table 10.6: The multiple-regression analysis for expertise coordination on security knowledge sharing.

Model 1	Unstandardized Coefficients		Standardized Coefficients			Collinearity Statistics	
	B	Std. Error	Beta	t	Sig.	Tolerance	VIF
(Constant)	2.418	0.217		9.878	000		
Coordinating organizational structure	0.151	0.085	0.128	1.768	0.078	0.414	2.416
Security infostructure	0.217	0.086	0.217	2.514	0.013	0.446	2.244

Dependent Variable: Security knowledge sharing.

10.5.3 Relationship between Security Knowledge Sharing and Learning Software Security

Table 10.7 indicates that security knowledge sharing has a significant positive correlation with software security learning in which the Pearson correlation is 0.578 and $p < 0.01$. The correlation coefficients of security knowledge sharing factors–codification knowledge sharing and personalization knowledge sharing are 0.491 and 0.455 respectively. The results showed a strong correlation among all security knowledge sharing factors that reached a significant level ($p < 0.01$). Thus, security knowledge sharing had a significant positive correlation with software security learning. Consequently, the research result favored hypothesis H_2: the stronger the codification and personalization knowledge sharing about software security, the higher the security learning level. Hence, H_{3a} and H_{3b} are also proven valid.

Tables 10.8 shows the result of the multiple regression analysis. As indicated in the table, B value, Beta, and *t*-value have positive values. The prediction equation is based on the unstandardized coefficients, as follows: $y_2 = 0.652 + 0.362x_5 + 0.216x_6$ (where x_5 is security knowledge sharing and x_6 is software security learning). All variables show a positive relationship. Looking at the *p*-value for each variable, we can see that the predictor variables of codification knowledge sharing and personalization

knowledge sharing are significant because both of their *p*-value is smaller than 0.05. This indicates that the regression model fits the data or there is a significant relationship between predictor variables (Codification knowledge sharing and Personalization knowledge sharing) and dependent variables (Software security learning). It also appears multicollinearity is not a concern because the VIF scores are both less than three. It shows a positive sign which indicates a positive linear relationship and the result is statistically significant. Thus, given the above relationship, hypotheses H_{3a} and H_{3b} are proven valid.

Table 10.7: The correlation analysis for security knowledge sharing and software security learning.

		Software Security Learning
Security knowledge sharing	Pearson correlation	0.578 **
	Sig. (2-tailed)	0.000
Codification knowledge sharing	Pearson correlation	0.491 **
	Sig. (2-tailed)	0.000
Personalization knowledge sharing	Pearson correlation	0.455 **
	Sig. (2-tailed)	0.000

** Correlation is significant at the 0.01 level (2-tailed).

Table 10.8: The multiple-regression analysis for security knowledge sharing on software security learning.

Model 2	**Unstandardized Coefficients**		**Standardized Coefficients**			**Collinearity Statistics**	
	B	Std. Error	Beta	t	Sig.	Tolerance	VIF
(Constant)	0.652	0.257		2.539	0.012		
Codification knowledge sharing	0.362	0.056	0.361	6.46	0.000	0.823	1.215
Personalization knowledge sharing	0.216	0.069	0.196	3.139	0.002	0.661	1.514

Dependent Variable: Software security learning.

10.6 Discussion

In this study, the research hypotheses are proposed with a conceptual framework, which was validated through conducting empirical examinations including survey question design, questionnaire data collection, validity and reliability testing, and correlation and linear regression analysis among 22 items in 324 valid questionnaires. The testing results of the research hypotheses are summarized in Table 10.9.

According to the result of the Pearson's correlation analysis (Table 10.4), there is a significant positive relation between security culture and security knowledge sharing. This means that if an OSS project truly holds the value that software security is

Table 10.9: Testing results of research hypotheses.

Hypothesis.	Result
H_1. *Security culture is positively associated with security knowledge sharing.*	Supported
H_2. *Expertise coordination is positively associated with security knowledge sharing.*	Supported
H_{2a}: *Coordinating organizational structure has a positive effect on security knowledge sharing.*	Partially supported
H_{2b}: *Infostructure has a positive effect on security knowledge sharing.*	Partially supported
H_3. *Security knowledge sharing is positively associated with software security learning.*	Supported
H_{3a}: *Codification knowledge sharing has a positive effect on software security learning.*	Supported
H_{3b}: *Personalization knowledge sharing has a positive effect on software security learning.*	Supported

important, then particular security knowledge sharing behaviors and actions can be expected. The more perceived normative support for security culture in their community means that participants are more likely to perform exemplary secure behaviors and avoid risk. As the security culture would certainly influence the operation activities of security knowledge sharing and further impact on the effectiveness of software security learning, the community should regard security culture as an important factor for supporting and guiding security practices.

Regarding the relation between expertise coordination and security knowledge sharing, this study finds that security expertise coordination is associated with the degree of security knowledge sharing. According to Pearson's correlation analysis (Table 10.5), there is a significant positive relation between security expertise coordination and security knowledge sharing. Moreover, when the factors of expertise coordination are more significant, they meaningfully affect security knowledge sharing, as evidenced by the significant variance explained by the regression analysis (Table 10.6). This implies that if the factors of expertise coordination—coordinating organizational structure and security infostructure are more efficient and effective—they can significantly enhance security knowledge sharing. Although the two factors do not have a significant correlation with security knowledge sharing in the regression model, they still have positive coefficients. Achieving a successful software system requires tight coordination among the various efforts involved in the software development cycle [252]. If OSS communities can provide an internal security consulting organization with dedicated responsible people for security activities, and place the security information in a structured and collected manner, it will lead to a knowledge-sharing arrangement actually being established.

On the other hand, our regression model also provides strong support for a significant contribution of security knowledge sharing to the software security learning process. The result of the Pearson's correlation analysis (Table 10.7) shows a significant positive relation between security knowledge sharing and software security learning. Moreover, as evidenced by the significant variance explained by the regression analysis (Table 10.8), while codification and personalization knowledge sharing is more significant, software security learning is significantly and positively affected. In the context of OSS communities, codification can be a good mechanism to store large amounts of security knowledge on the project website and to create an organizational memory for all participants. The method of personalization knowledge sharing reflects security experts' experience (via the forum, mailing list, code review, etc.) which collectively produces knowledge that can be spread further to the individuals or the whole team. The two knowledge-sharing mechanisms create a digital pipeline or an intelligent link for knowledge building that appears to support the software security learning process. As the community provides opportunities for its members to share security knowledge or experiences with others, which increases the amount of knowledge sharing, it should stimulate software security learning.

10.7 Conclusions

This empirical study focuses on investigating the organizational practices and behaviors that affect knowledge sharing and learning about software security in OSS communities, and the relationships among them. OSS has become a critical component and a key competency of information and communication technology (ICT) ecosystems. While the number of found vulnerabilities in OSS is increasing, it is noteworthy that effective learning about security knowledge in the context of OSS development has not gained much attention. Thus, it is necessary to examine how the security knowledge is transferred and acquired by OSS participants.

As Scacchi points out, the meaning of open source in the socio-technical context is broader than its technical definition and includes communities of programming practice, organizational culture and structure, and technical practices [390]. This can be viewed as a necessary condition within a learning framework as both social and technical aspects are of equal importance. This research proposes a model that helps conceptualize the linkage between such socio-technical practices and the software security learning process in OSS communities. We gathered empirical evidence from 324 questionnaires and quantitatively analyzed data to test the hypothesized relationships in the model.

The statistical analysis shows that both security culture and the coordination of expertise can positively influence and contribute to security knowledge sharing at a certain level in OSS communities. Security culture provides a strong indication of a participant's disposition to act. It is important because unless the community believes that security is valuable to the software product, participants are unlikely to work

securely and exchange their experiences in the field of software security. Indeed, every member involves in OSS development should be concerned with software security, but it is inefficient to demand each participant taking care of all security aspects. Hence, in order to enhance security knowledge sharing, a community should cultivate a culture that engages dialogue and interest among participants in order to promote the value of software security to their products and raise awareness. If OSS communities can nurture a security culture, it will be easy for them to create an environment where developers and users are willing to share and talk about software security, providing the opportunity to draw lessons from each other's experiences.

On the other hand, as OSS and its communities continue to grow in size and complexity, security expertise coordination within the community plays a larger role in security governance. While security information is provided with an adequate coordinating structure and infostructure support in the community, the implementation of security knowledge sharing throughout the community can be instilled in its culture. This study also concludes that the learning process (self-directed and collective learning) of software security and learning satisfaction is definitely influenced by security knowledge sharing. It indicates that the successful sharing of security knowledge in the OSS community, either through codification or personalization mechanisms, will enable software security learning to flow through an entire community.

People join the OSS community at different ages and have different backgrounds, capacities, and resources, as well as different objectives. They come from many disciplines that might lack formal, college-level software security training, and therefore do not see any economic incentive for squeezing security thinking into their work to produce secure codes. On the other hand, learning software security is a difficult and challenging task as the domain is rather context-specific, and the real project situation is necessary to apply the security concepts within the specific system. It is suggested that OSS communities must establish beliefs and norms, as well as roles and knowledge facilities for secure software developments; i.e., to offer environments and opportunities for security knowledge sharing and the development of software security knowledge for participants as well on the horizontal level between the experienced (but ever-learning) community members.

Ultimately, the contributions of this research supply researchers with a conceptual framework for software security knowledge sharing and learning in the OSS community in a thorough manner, providing a context in which to operate. The study also provides other researchers with a firm basis to develop new security learning approaches for OSS communities, addressing many of the identified limitations.

10.8 Limitations

Several limitations of this research should be noted. Despite a rigorous examination of the trustworthiness of the collected data, this study might have some method bias.

First, the samples were chosen opportunistically from GitHub projects, and the number of responses obtained from the survey was rather small compared with today's enormous OSS projects and field workers. Second, even though there are other known human factors that facilitate security knowledge sharing behaviors in organizations as Safa and Von Solms suggested, this study did not consider factors such as motivation or intention in OSS communities [385]. Thus, there is a need for further research efforts focused on accumulating more evidence that is empirical and data to break through the limitations. These efforts should improve the generalizability of this study to the entire OSS development phenomenon by considering a larger number of responses covering a range of diverse OSS projects. In addition, special attention should be geared toward finding the human factors, which affect independent variables such as reputation, self-efficacy, and promotion.

Chapter 11

Towards a Context-Based Approach for Software Security Learning

Wen, Shao-Fang and Katt, Basel. "Towards a Context-Based Approach for Software Security Learning." *Journal of Applied Security Research*. 2019, volume 14, issue 3, pp. 288-307.

Author Contributions— Initial conceptualization and framework of the research were developed by Shao-Fang Wen. The research methodology and experimental design and were reviewed by Basel Katt. The manuscript was largely written by Shao-Fang Wen.

Abstract— Learning software security is one of the most challenging tasks in the information technology sector due to the vast amount of security knowledge and the difficulties in understanding its practical applications. Conventional teaching approaches give little attention to how to improve the effectiveness of learning in the domain of software security. Context-based learning has been proven to be a sound pedagogical methodology; however, it is still unclear how to synthesize the prescription in the domain of software security. In this paper, a context-based approach to software security learning is proposed for structuring and presenting security knowledge. To evaluate the proposed approach, a quasi-experiment was designed and executed in the setting of a university learning environment. The experiment results indicate that the proposed context-based learning approach not only yields significant knowledge gains compared to the conventional approach. but also gains better learning satisfaction of students

11.1 Introduction

Information technology is one of the world's fastest-growing industries. In fact, the rate at which software and software products are evolving is many times greater than the rate at which software security is evolving. According to CVE vulnerability data [102], the number of software vulnerabilities disclosed in 2017 grew by 128% compared to the number in 2016, reaching an all-time high of 14,714. In an age of cybercrime, some of the most widespread software-based crimes include stealing information via hacking, carrying out virus attacks to take down computer systems and implanting spyware with the intent to watch a person or his or her computer activities. Due to the increasing importance and complexity of computer systems, insufficient knowledge and skills related to software security will result in more serious breaches in the future.

Software security knowledge is multifaceted and can be applied in diverse ways [294]. Learning software security is a complex and difficult task because learners must not only deal with a vast amount of knowledge about a variety of concepts and methods but also have to demonstrate the applicability of the knowledge through experience in order to understand their practical use. Conventional security learning materials are usually subject-oriented, which is useful for rote memorization of a specific subject or for information recall later. However, such an approach makes it difficult for learners to understand the rationale of the topics and correlate those topics with real software cases. Learners often feel that security knowledge is so extensive and software security is so difficult to achieve that they simply cast it aside [23].

In traditional software security teaching, little attention is given to what the security knowledge really means to learners, and there is not much content addressing the connections between real-world situations and security concepts. According to Jonassen and Land [224], "...learners must be introduced to the context of the problem and its relevance, and this must be done in a way which challenges and engages them. Context and the particulars of that context can provide a powerful motivation for learning"[352]. If learners do not learn the knowledge well in the first place, they cannot possibly transfer it to new situations [90]. We argue that, in order to regulate learning about software security effectively, security knowledge should be contextualized and embedded in a meaningful scenario that makes sense to the students to enhance their understanding and make the concepts more relatable.

The introduction of context in security education attempts to bridge the gap between abstract concepts and everyday life, in order to show students or security learners the relevance of science for their own lives and interests and to improve their motivation for learning about security content. The concept of learning in context has been widely addressed in education and psychology literature over the years, and the effectiveness of context-based learning has been demonstrated in the setting of

interactive school classrooms. However, it is still unclear how this concept can be synthesized and applied in the domain of software security. To mitigate this research gap, we proposed a context-based approach to structure security knowledge and facilitate software security learning in a way that can motivate learners. We conducted experiments to evaluate the effectiveness of this approach in the setting of a university learning environment. This paper presents the rationale of the proposed approach and the findings of our experimental studies.

11.2 Conventional Security Learning Materials

In conventional security learning materials, the knowledge content is commonly organized topically, focusing on security aspects. One approach may first introduce attack patterns or security vulnerabilities (the black-hat side), such as Cross-Site Scripting (XSS) and SQL injection (SQLi), while another might start with secure design practices or coding standards (the white-hat side), such as input validation and output encoding. The security-centric materials are often written in the form of a reference manual or a guide to a particular security certification. Learners usually finish reading such materials with little understanding of the context in which the security knowledge should be applied. This relates to what is known as the knowing-doing gap; that is, knowing better but not doing better.

On the other hand, security learning materials usually emphasize concepts first rather than facts or context to transmit knowledge. Consequently, learners may struggle to finish reading them due to a learning style mismatch. Several studies [136, 291, 292] have shown that the majority of engineering students are sensor-type learners, who like facts, data, and observable phenomena as opposed to theoretical abstractions. Since many security tasks require awareness of one's surrounds, attentiveness to detail, experimental thoroughness, and practicality, the learning material presented must provide meaning and motivation for learners, allowing them to learn security principles and processes through a real-world situation that is of particular interest to them.

11.3 General Concepts of Context-Based Knowledge for Learning

According to Oxford Dictionaries[37], context is defined as "The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood." Meanwhile, Dey [117] defined context as "a set of information used to characterize the situation of an entity." Nonaka and Konno [328] noted that knowledge reflects a particular stance, perspective, or intention in accordance with the characteristics of a specific context, which is different from information. Knowledge comes from a variety of contexts, and it cannot be accurately understood

[37] https://en.oxforddictionaries.com/definition/context

without context [58, 242]. Without proper contextual information, knowledge can be isolated from other relevant knowledge, resulting in limited or distorted understanding [61, 169]. Since context can provide guidance regarding when, where, and why a piece of knowledge is used, it is crucial to consider the context to enhance the applicability of the knowledge.

Context can increase the information content of natural language utterances and facilitate learning [57, 59]. Psychology and education researchers have demonstrated that when knowledge is learned in a context similar to that in which the skills will actually be needed, the application of the learning to the new context may be more likely [117, 122, 352]. Predmore [360] showed that learning about knowledge content through real-world experience is important because "once [students] can see the real-world relevance of what they're learning, they become interested and motivated." The book How People Learn [90] also pointed out that motivation is critical for learning, enabling knowledge transfer to occur. If students do not learn the material well in the first place, they cannot possibly transfer it to new situations. As stated in the book "Learners of all ages are more motivated when they can see the usefulness of what they are learning and when they can use that information to do something that has an impact on others" [90] (page 49).

Bennett and Lubben [38] offered a definition of a context-based approach to science education: "Context-based approaches are approaches adopted in science teaching where contexts and applications of science are used as the starting point for the development of scientific ideas." The authors reported that context-based science courses motivate students and help them become more positive about science by representing real-world situations of the learning subject. When students are more interested and motivated by the experiences they are having in their lessons, their increased engagement may result in improved learning [38]. In computer science education, there is also a broad agreement that teaching units should start from a "real-world" context or phenomenon, aiming to create connections to prior knowledge, increase the relevance of the material to students, or show applications of the intended knowledge, thereby increasing motivation [120, 184]. These contrast with more traditional approaches that cover abstract ideas first, before looking at practical applications.

Likewise, in software engineering, studying in one context and then abstracting the knowledge gained for use in a new context is a common way of learning programming that has been observed extensively in both new and experienced programmers [23, 243]. In order to capture and use security knowledge appropriately, it is necessary to first specify which context information is to be handled. Then, it must be represented in a format that is understandable and acceptable to the individuals. Thus, a context for a software security topic includes the circumstances in which its technical content exists. Therefore, when talking about software security in a given context, the knowledge would not only include the basic principles and

processes of software security but also consider how security knowledge is used in one or more particular domains or application areas.

11.4 The Proposed Context-Based Approach

To facilitate contextual learning about software security, we proposed a context-based approach to structuring and presenting software security knowledge using three strategies: (1) Using a meaningful application scenario; (2) Simulating learners' mental models for security learning, and (3) Moving from concrete to abstract security knowledge. Figure 11.1 shows the conceptual view of the proposed context-based learning approach with three strategies.

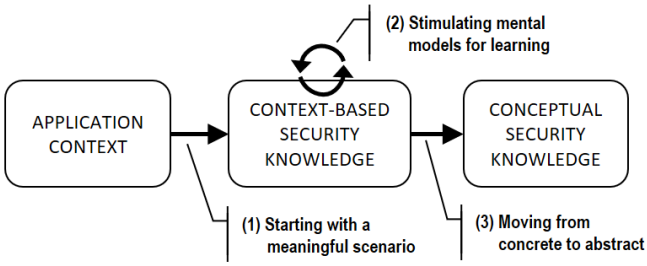


Figure 11.1: A conceptual representation of the proposed learning approach for software security

11.4.2 Starting with a Meaningful Scenario

Contextualized learning often takes the form of real-world examples of problems that are meaningful to the learners personally [373]. To begin the process of learning, a meaningful situation for learners must first be established. In our approach, we set the application context as the starting point for learning security concepts on a need-to-know basis. Figure 11.2 presents the main components of the application context, which include application paradigms, application functionalities, and application scenarios. The application paradigm is a combination of security-independent data that characterize software applications; for example, the domain area that the application belongs to or the technologies that the application uses. The software functionality represents any aspect of software applications that can perform for users or other systems in a particular paradigm, such as dynamically generating HTML in web applications and cleartext transmission of sensitive information in network applications. Under a given application paradigm and functionality, a series of scenarios are identified, each of which deals with one specific scenario in the context.

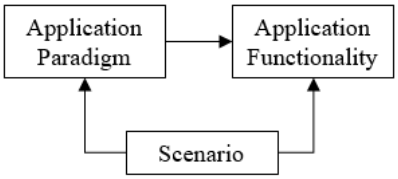


Figure 11.2: Components of the application context

A scenario is made up of practical demonstrations of the pre-described application functionality and the code fragments behind it that bridge the corresponding security knowledge. In this manner, a scenario constitutes a form of an anchoring event [85], which provides an experiential practice in software development from which learners can relate to new information about the security. Research has shown that using anchoring events in learning promotes memory recall and the subsequent transfer of information to a new setting [85], which helps to render abstract ideas more concretely and thus provides a cognitive mooring around which newly learned ideas can be linked with learners' prior understandings [86]. When learners see applications and software function with the code they are already familiar with, (i.e., the anchor event), the consequence of exploiting vulnerabilities hits close to them and becomes more real, further motivating them to learn.

11.4.3 Stimulating Mental Models for Learning

In order to help learners create a strong and lasting bond that makes navigating the security knowledge efficient, we developed a knowledge structure to guide them in approaching personal mental models in the software security domain. Mental models combine a schema or a knowledge structure with a process for manipulating the information in the memory [304], while knowledge structure interrelates a collection of facts or concepts about a particular topic. Craik [94] suggested that the human mind builds and constructs "small-scale models" to anticipate events. Such mental models allow learners to gain insight regarding their world by building a work scheme [160], which makes it easier for them to access the information needed to understand the knowledge domain, make predictions, and decide upon action to take [379]. This can result in successful learning by engaging students, fostering their concentration, and assisting them in organizing systemic information [402].

To design a security knowledge structure (schema) that is easier to store in the learners' memory, we simplified the schema and reduce the content load of the knowledge structure. We identified the critical security concepts that are most widely used throughout the security domain and concentrated learning approaches on them. Ultimately, three security concepts were incorporated into the knowledge structure: security attack, security weakness, and security practice. Table 11.1 provides the definitions of the three security concepts. Generally, our intention was to guide learners in answering three questions while dealing with each scenario:

- What are the possible attacks?

- Why does it encounter attacks?
- How can these attacks be prevented?

Table 11.1: The definition of security concepts and the corresponding focus questions

Security concept	Definition
Security Attack	It represents actions taken against the software case with the intention of doing harm.
Security Weakness	It represents bugs, flaws, vulnerabilities, and other errors in the software case.
Security Practice	It represents methods or mechanisms to mitigate security weaknesses to prevent security attacks.

Figure 11.3 illustrates the relationships between the concepts embedded in the proposed knowledge structure in the domain of software security. The knowledge structure provides the basis for the development of mental models in learning software security knowledge. As learners answer the what–why–how questions for each scenario, the relationships between the security concepts are emerging in their midst, and thus, their mental model expands.

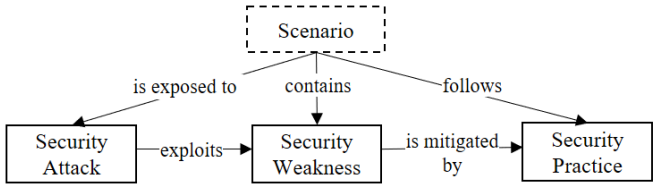


Figure 11.3: The relationship among security concepts of the knowledge structure

11.4.4 Moving from Concrete to Abstract Security Knowledge

Security Knowledge can be categorized as concrete or abstract facts, events, applications, conceptual descriptions, and principles. To help learners gain a more flexible understanding of the study concept in a range of situations with varying levels of abstraction, we organize security knowledge by blending abstract and concrete perspectives; presenting it with a sequence from concrete to abstract. In our study, abstract knowledge refers to the conceptual security domain knowledge while concrete knowledge relates to the contextualized scenario-specific security knowledge. Research has shown that presenting knowledge in both concrete and abstract terms are far more powerful than presenting either one in isolation [348]. Lave and Wenger [264] also argued that abstract and generalized knowledge gains its power through the expert's ability to apply it in specific situations.

The used concrete-to-abstract approach in knowledge presentation differs from the traditional, where the concepts are of foremost importance and are usually explained

first before concrete examples and applications are discussed. Figure 11.4 depicts the learning paths that are constructed by the proposed context-based approach. In such a concrete-to-abstract knowledge presentation, learners discover meaningful relationships between practical functions and abstract knowledge in the context of real applications. The value of concrete representations has been frequently noted in education. Concrete materials can support abstract reasoning because they can be explicitly designed to promote true inferences from perceptual representations to abstract principles [35]. A method known as concreteness fading [170] has the advantage of initially presenting concepts in a concrete fashion and then, over time, augmenting that initial presentation with progressively more abstract representations of the concepts. Abstract understanding is most effectively achieved through experience with perceptually rich, concrete representations [171], while concrete materials make concepts real and therefore easily internalized [226]. As long as the concrete knowledge and the underlying abstract explanation are understood by learners, learning transfers from one context to another will be more effective.

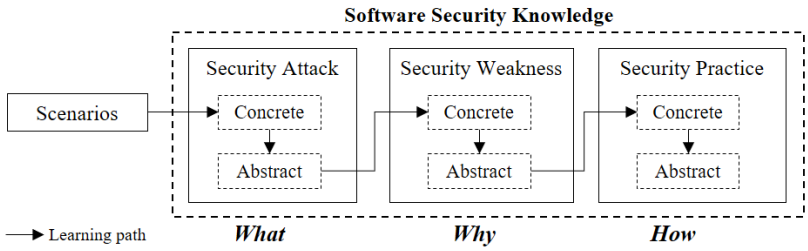


Figure 11.4: The constructed learning path based on the context-based approach

11.5 Study Method

To evaluate the proposed approach, a quasi-experiment with non-equivalent groups was designed and executed in the setting of a university learning environment. Our hypothesis in this study was:

Hypothesis: *The context-based approach to supporting students' software security learning yields better knowledge gain and learning satisfaction than the conventional learning approach.*

Two rounds of experiments with learning subjects related to Web Security were conducted with Bachelor students; each round lasted for about 70 minutes. According to the hypothesis, the variables in this experiment were defined as followings:

- **Independent variables:** The learning approaches (i.e., conventional vs. contextualized).
- **Dependent variable:** The security knowledge gain and learning satisfaction were measures providing insight into the effectiveness of the two approaches.

In this section, the sources of data, the tools used for data collection, the participants, and the experimental procedure are briefly outlined.

11.5.1 Participants

The participants were 42 Bachelor students from the fifth semester (third year), who were taking the "Software Security" course. The students were from two main study programs: Bachelor in IT Operations Information Security and Bachelor in Programming.

11.5.2 Treatments

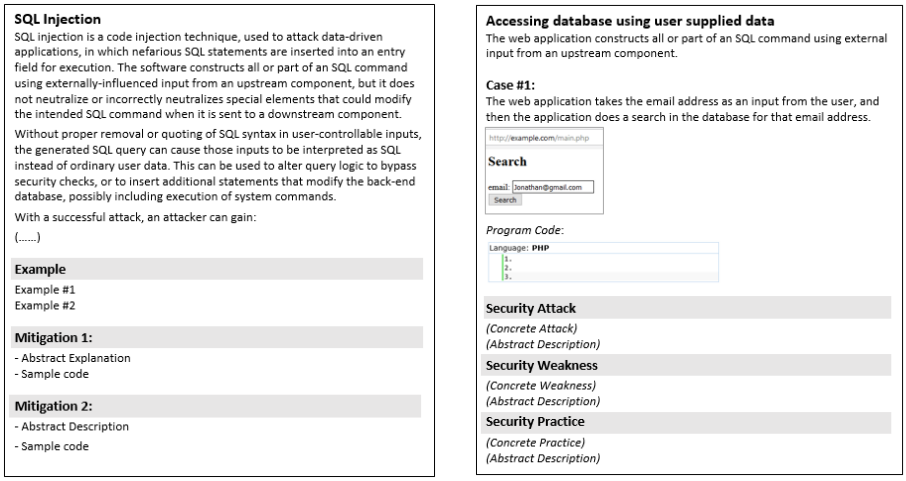
In this study, we designed two types of learning materials in a printed format as the experimental treatments, which were named type I and type II. The type I material used a conventional approach while type II adopted the proposed context-based approach to organizing software security knowledge. Regarding the learning subject, we used two common software vulnerabilities in web applications: SQLi and XSS. The materials were constructed using resources on the internet (e.g., OWASP and CWE) combined with the authors' teaching experience in the domain of software security. In the type I material, information was presented in the order of abstract to concrete. Conceptual knowledge about the vulnerability subject was described first, followed by examples with code fragments. Mitigations for the vulnerabilities were explained in the last section.

For the construction of the type II learning materials, we first set up the learning environment in a web application paradigm—an e-Store—using the LAMP38 web service stack. For this specified context, the author developed a preliminary set of functionalities to operate a web-based e-Store application, including a login module, data input/output features, data processing, database access, and payment functions. Three critical application scenarios were created for each of the learning subjects within the scope of the e-Store functionalities. In the learning materials, functional features with the corresponding code fragments for each scenario were described and demonstrated in the beginning, followed by the security knowledge, which was organized based on the predefined knowledge structure (i.e., security attack, security weakness, and then security practice). Knowledge content for each security concept was presented in the order of concrete to abstract. All content demonstrating concrete knowledge was manipulated using the built application, including coding vulnerabilities, exploits, and code fixes.

Figure 11.5 shows the simplified view of the two types of learning materials in the subject of SQLi vulnerability. In terms of the type II material, three scenarios were

[38] LAMP is an open source web service stack that uses Linux as the operating system, Apache as the web server, MySQL as the relational database management system, and PHP as the object-oriented scripting language.

introduced under an abstract functionality, "Accessing database using user-supplied data," which formed as the anchoring event for subsequently studying relevant security knowledge.



(a) Type I material (b) Type II material

Figure 11.5: The simplified view of two learning materials for SQLi vulnerability

11.5.3 Data Collection

To collect data and measure the dependent variables, two types of instruments were used: pre- and post-tests and survey questionnaires. Pre and post-test sheets were developed to measure the learning gain (post-test/pre-test), in which items were created covering two types of security knowledge: theoretical and practical. The theoretical items focused on recalling and understanding conceptual security knowledge. The practical items required students to identify possible attacks in a given software context, mark coding errors in code fragments, and apply knowledge to different situations. The pre- and post-tests were similar except for the formulation of some questions, their order, and the answer options. Four test sheets (pre- and post-test for two rounds) were generated to assess the students' level of knowledge before and after the learning sessions. In each test sheet, there were 10 questions (6 theoretical and 4 practical), and the value for each question was five points.

We designed a survey questionnaire to collect students' perceptions of the two learning materials. Students were asked six questions for each type of learning material, which we used to measure the learning satisfaction factors, including interest creation, content fulfillment, learning efficiency, experience correlation, positive attitude, and personal satisfaction (Table 11.2). In this questionnaire, all respondents were required to choose the answer that reflected their own views and stance on the statements that were administered in accordance with a 5-point Likert scale, ranging from "strongly disagree" to "strongly agree."

Table 11.2: Questionnaire items for measuring learning satisfaction

Factor	Question
Interest Creation	I feel that the material is interesting when I get into it.
Content Fulfillment	The material provides knowledge content that fits my needs precisely.
Learning Efficiency	The material helps me learn secure programming efficiently.
Experience Correlation	I could relate what I learned from the material to what I have already known or experienced before.
Positive Attitude	The material helps me foster a positive attitude towards learning about secure programming.
Personal Satisfaction	I find that at times studying the material gives me a feeling of personal satisfaction.

11.5.4 Experimental Procedure

The students were divided into two groups (group A and group B) after being seated in the classroom. They were first introduced to the main objectives of the experiment and informed of the procedure. Both rounds of experiments were performed with a similar experimental procedure. Table 11.3 shows the learning subject arrangement and the dispatch rule of learning materials in each round/group. In the first round, students were given test sheets (pre- and post-test) and learning materials for the subject of SQLi. Students in group A studied type I learning the material, while group B studied type II material. In round 2, the learning subject was changed to XSS, and we switched the type of learning material treated in the two groups. With the two-round experiment design, all students were able to experience both learning materials and thus the differences between the two. The major experiment steps in each round were as follows:

Step 1: Pre-test (15 minutes)

Step 2: Learning session (40 minutes)

Step 3: Post-test (15 minutes)

There was a 10-minute break between the two rounds. At the end of the second round, students completed the learning satisfaction questionnaire. This ended the experimental procedure.

Table 11.3: Learning materials dispatching rules

	Treatment	
	Round 1 (SQLi)	Round 2 (XSS)
Group A	Type I	Type II
Group B	Type II	Type I

11.6 Findings

In this section, we present the findings of the experiment, including an evaluation of the students' knowledge gain and learning satisfaction.

11.6.1 Knowledge Gain

The students' knowledge gain in the different types of materials was determined using a comparative means analysis. Table 11.4 presents the means analysis of the students' performance on the pre- and post-tests in each round of the experiment, including the mean scores and standard deviations. The results of the statistical analysis show that there was a positive knowledge gain (i.e., post-test to pre-test score) for both groups in both rounds. However, the group using type II materials had higher achievement levels than the group using type I materials, as shown in Figure 11.6.

To determine whether there was a significant difference between the pre-test performances of group A and group B, an independent sample t-test was used. Table 11.5 shows the t-test analysis for the pre-test means scores in the first round. The significance level (0.628) of Levine's test for equal variance was greater than 0.05, indicating "Equal variance assumed." Levine's test resulted in a "Sig. (2-tailed)" value of 0.137, which was above 0.05. Therefore, the null hypothesis of the independent sample t-test was rejected ($p > 0.05$), which implies that there were no significant differences between the two groups in terms of pre-test scores (i.e., the initial security knowledge) so that the significance of the knowledge gain can be concluded).

Table 11.6 shows the independent sample t-test results in the first round for the post-test mean scores. Moreover, the difference between the post-test mean scores of the two groups is significant (2-tailed sig. = 0.02, $p < 0.05$). This indicates that our treatments resulted in a significant difference in security knowledge gain in the two groups of students.

Table 11.4: Comparative means analysis of students' performance on the pre- and post-tests

		Group A			Group B		
Round		N	Mean	SD	N	Mean	SD
1	Pre-test	20	26.75	5.20	22	24.32	5.19
	Post-test	20	29.50	6.90	22	33.86	4.86
2	Pre-test	20	21.75	8.78	22	20.00	9.26
	Post-test	20	26.25	6.90	22	30.91	8.54

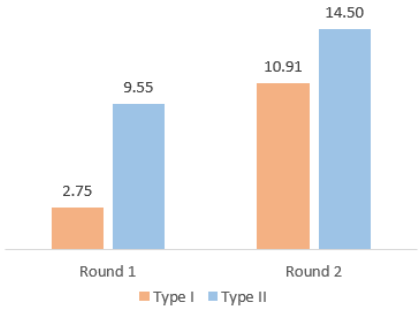


Figure 11.6: Knowledge gain for the two groups in each round of experiments

Table 11.5: Independent sample t-test results for pre-test scores (1st round)

		Levine's Test		t-test				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Pre-Test	Equal variances assumed	0.238	0.628	1.516	40	0.137	2.432	1.604
	Equal variances not assumed			1.516	39.601	0.138	2.432	1.605

Table 11.6: Independent sample t-test results for the post-test scores (1st round)

		Levine's Test		t-test				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Post-Test	Equal variances assumed	2.415	0.128	-2.413	40	0.020	-4.414	1.829
	Equal variances not assumed			-2.374	33.793	0.023	-4.414	1.859

We performed the same statistical analysis for the pre- and post-test scores in round 2 (Table 11.7). As can be seen in Table 11.7, there was also no significant difference in the pre-test scores in the two groups (2-tailed Sig. = 0.534, $p > 0.05$). The post-test 2-tailed Sig. was 0.032, thus achieving significant and indicating that the post-test score would also be affected by treatments in round 2.

Table 11.7: Independent sample t-test for pre- and post-test score (2nd round)

		Levene's Test		t-test				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Pre-Test	Equal variances assumed	0.012	0.913	0.627	40	0.534	1.750	2.791
	Equal variances not assumed			0.629	39.921	0.533	1.750	2.784
Post-Test	Equal variances assumed	0.063	0.802	2.220	40	0.032	5.341	2.406
	Equal variances not assumed			2.243	39.431	0.031	5.341	2.381

11.6.2 Learning Satisfaction

The learning satisfaction for the two learning materials is represented as a radar chart with six axes (Figure 11.7). As depicted in the chart, the type II material had overall higher learning satisfaction mean scores than the type I materials in terms of the six satisfaction factors. Regarding the data series of the type II materials, the score of the six satisfaction factors were all above 4. Almost all of the responses regarding the type II were at least 3, and responses of 1 and 2 were rare. Of these, the mean scores of "Interest Creation" and "Experience Correlation" were the highest (4.33 and 4.29, respectively). In contrast, the scores of the two factors in the type I materials had the lowest mean scores (i.e., 2.81 and 2.83, respectively). The mean scores of the four other satisfaction factors evaluated for the type I materials were all approximately the same (3).

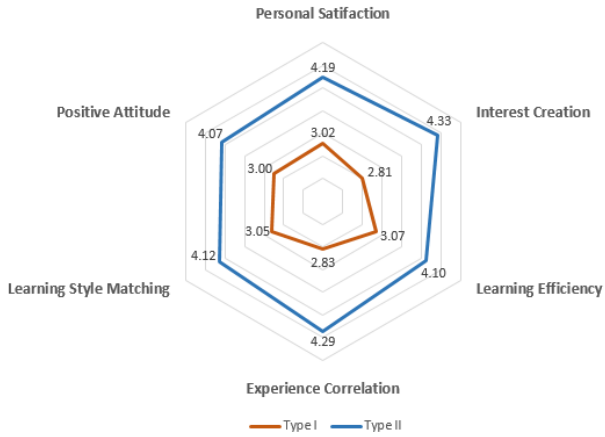


Figure 11.7: Radar diagram for learning satisfaction scores

11.6.3 Additional Findings

In this study, we were also interested in how the students performed with theoretical and practical questions when they were presented with type II learning materials. According to Table 11.8, students performed better in the pre-test on theoretical questions than on practical ones in terms of hit rate (overall hit rate: 54.70% vs. 33.13%). After the type II materials were presented there was a knowledge gain in either the theoretical or practical questions. The average hit rates of both categories in the post-test reached the same level. In the first round, they fell to between 65% and 70%, while they were between 70% and 75% in the second round. Regarding the growth ratio of the mean scores from the pre-test to the post-test, it is clear that the students had better achievement with practical questions (110.29%) than with theoretical questions (28.74%).

Table 11.8: Comparative means of students' performance on the pre- and post-tests

		Pre-test			Post-test			Growth
Round		N	Mean	Hit Rate	N	Mean	Hit Rate	Ratio
1	Theoretical	6	16.82	56.06%	6	20.00	66.67%	18.92%
	Practical	4	7.50	37.50%	4	13.86	69.32%	84.85%
2	Theoretical	6	16.00	53.33%	6	22.25	74.17%	39.06%
	Practical	4	5.75	28.75%	4	14.00	70.00%	143.48%
Sum	Theoretical	12	32.82	54.70%	12	42.25	70.42%	28.74%
	Practical	8	13.25	33.13%	8	27.86	69.66%	110.29%

11.7 Discussion

The objective of this study was to evaluate a context-based approach to improving learning about software security. A two-round pre-test/post-test experiment was used to measure the students' security knowledge gain, and a questionnaire was used to evaluate their learning satisfaction. The results of the pre-test/post-test experiment indicate an increase in the students' level of security knowledge for both the conventional and context-based approaches. According to the statistical t-test analysis, there was no significant difference between the two groups in terms of initial security knowledge; however, students using treatments with the context-based approach had significantly better knowledge gain than those using treatments with the conventional approach. The evaluation of the students' satisfaction with the two learning approaches supports our hypothesis, as the respondents showed higher learning satisfaction with the context-based knowledge approach than with conventional approaches.

As highlighted by the learning satisfaction analysis, a majority of students using conventional materials were unable to make connections between what they were learning about security and what they had been doing in programming. We argue that the way they process information and their motivation for learning is not supported by the conventional methods. Research has indicated that learning is most efficient when it is linked with the experience and prior knowledge that students bring to a given learning situation [90, 266]; however, novice learners do not always make connections between new information and prior knowledge or everyday experiences in ways that are productive for learning [259]. In the context of software security learning, learners interpret the security knowledge they gain through a range of strongly held personal programming experiences. They often do not associate vulnerabilities with programs similar to what they were writing previously. Therefore, establishing the relevance of learning materials before going into the details could provide a concrete foundation for the learning process.

Our approach attempts to place security learning in the context of real application scenarios, which serve as anchoring events and elicit the learners' memories and draw attention to software events and conditions. The results of our experiment show that this type of design keeps learners interested, motivated, and engaged in the learning experience. Since the given context is connected and relevant to their prior knowledge and life experiences in software development, security learning can then be related to a similar programming topic that they want to learn about or a problem to be solved. According to the results of the learning satisfaction survey, most students were very interested in studying type II materials and agreed that the materials could be correlated with their experiences. We believe this implies a direct effect on higher overall learning satisfaction, which motivates students to learn. The benefits of the contextualized approach can also be explained by the effective mechanism of intrinsic motivation, where a learner is drawn to engage in a task because it is perceived as interesting, enjoyable, and/or useful [89, 115, 251].

In this study, we investigated how the contextualized approach affects students' learning performance in terms of answering theoretical and practical questions. The results show that type II materials can effectively support both abstract and concrete learning, and moreover, they provide a greater influence in terms of dealing with practical problems. Hence, a blend of concrete and abstract knowledge presentation can help learners gain a more flexible understanding of the study concept in a range of situations with varying levels of abstraction. Research has shown that presenting knowledge in both concrete and abstract terms are far more powerful than presenting either one in isolation [348]. Deductive reasoning is facilitated when the domain is familiar and concrete rather than abstract [476]. Our approach begins with the presentation of concrete information in a context familiar to students, which gradually leads to an abstract understanding. As long as the concrete knowledge and the underlying abstract explanation are understood by learners for a specific situation, learning transfers from one software context to another will be more effective.

11.8 Conclusion

In this paper, a context-based approach to presenting security knowledge is proposed for software security learning. This approach is composed of three main strategies. The first is to establish an application context to create a meaningful situation for learners, which is described by application domains, application functionalities, and scenarios. The design of the application context aims to activate the learner's prior knowledge of software programming and anchors the learning about security knowledge. The second strategy is to organize underlying security knowledge in a structured manner that can stimulate learners' mental models to support more efficient learning in the specified context. The third is to guide learners to engage with concrete knowledge before studying abstract knowledge. This strategy assists learners in discovering meaningful concepts and relationships between practical

functions and abstract knowledge when working in this context. Furthermore, it helps them apply knowledge in various other contexts.

The approach was evaluated through a controlled quasi-experiment with 42 Bachelor students. There were positive findings in terms of security knowledge gain and learning satisfaction when students studied learning materials that were constructed using the context-based approach. According to the results, the proposed approach provides a sounder basis for software security learning than conventional methods. It is recommended that curriculum developers of software security courses should use the context-based approach as one of the teaching strategies to improve students' performance in security knowledge learning. In the future, we plan to promote this approach for teaching secure programming and to use it to build a web-based learning application. We believe that such an online learning environment would allow more learners' to benefit from the learning approach.

Chapter 12

Managing Software Security Knowledge in Context: An Ontology-Based Approach

Wen, Shao-Fang and Katt, Basel. "Managing Software Security Knowledge in Context: An Ontology-Based Approach." *Information*, 2018, volume 10, issue 6.

Author Contributions— Initial conceptualization and framework of the research were developed by Shao-Fang Wen. The research methodology and experimental design and were reviewed by Basel Katt. The manuscript was largely written by Shao-Fang Wen. Final paper review and editing were performed by Basel Katt.

Abstract— The knowledge of software security is highly complex. To secure software development, software developers require not only knowledge about the general security concepts but also about the context for which software is being developed. With traditional security-centric knowledge formats, it is difficult for developers or knowledge users to retrieve their required security information based on the requirements of software products and their used technologies. In order to effectively regulate the operation of security knowledge and be an essential part of practical software development practices, we argue that security knowledge must specify contextual characteristics needed to be handled, and represent the security knowledge in a format that is understandable and acceptable to the individuals. This paper introduces a novel ontology approach for modeling security knowledge with a context-based approach, by which security knowledge can be retrieved taking the context of the software application in hand into consideration.

12.1 Introduction

The knowledge of software security is highly complex since it is quite context-specific and can be applied in diverse ways [294]. Software developers not only require knowledge about the general security concepts but also need the expertise to deal with variant technologies, frameworks, and libraries that are involved with software development projects [381]. The complex security knowledge usually surpasses the capacity of software developers to solve security problems by themselves [199]. For example, the security principle of least privilege recommends that accounts should have the least amount of privilege required to perform the task. This encompasses security practices of user rights, and resource permission such as CPU, memory, and network, which exist with different programming languages (e.g., C, C++, PHP, Java and so on), depending on the features of software products. However, much of the required security knowledge is traditionally encapsulated in unstructured or semi-structured formats [78] and commonly organized in a security-centric structure, as either back-hat or white-hat security. With such topical security knowledge formats, it is difficult for developers or knowledge users to retrieve the required security information based on the requirements of software products and the used development technologies.

Therefore, in order to effectively operate security knowledge and be an indispensable part of practical software development practices, we argue that security knowledge must first incorporate additional contextual features, that is, to contextualize security knowledge with certain characteristics of software applications, and then represent it in a format that is understandable and acceptable to the individuals. Ontology has been regarded as a good knowledge management approach in the domain of information security to methodically classifying various security concepts, such as security attacks and vulnerabilities as well as related security prevention mechanisms [143, 443]. The knowledge representation of ontology not only integrates knowledge resources at both abstraction and semantic levels, but can also be adopted by knowledge sharing services such as advanced knowledge search, knowledge visualization, and therefore, supporting the learning process of software security.

This paper is part of ongoing research on developing a contextual learning environment for software security, in which an ontology is used as the kernel knowledge repository in managing contextualized security knowledge. The objective of this research work is to support software developers and knowledge users to define and use security knowledge appropriately, adapting to their working context. The ontology we designed integrates application context, security domain knowledge, and contextualized knowledge, allowing contextual inquiry through software scenarios that users would be interested in or familiar with. In this paper, we present our security ontology with the design concepts and the evaluation process.

The rest of this paper is organized as follows. Section 12.2 introduces background knowledge about the context and knowledge. In Section 12.3, we describe the design

of our ontology. Section 12.4 presents the evaluation process of the ontology, followed by a discussion in Section 12.5. We discuss related work in section 12.6. Lastly, Section 12.7 presents the conclusion and our future works.

12.2 Context and Knowledge Management

According to Brézillon [59], ''context is a set of information used to characterize a situation in which human and computational agents interact". He also points out that, knowledge comes from a variety of context and it cannot be accurately understood without context [58, 61]. The context has the capacity to provide a major meaning to knowledge, promoting a more effective comprehension of a determined situation in the collaborative work [60]. Context is a crucial component of a full understanding of knowledge [58, 219, 242]. Without appropriate contextual description, knowledge could be isolated from other relevant knowledge, resulting in limited or distorted understanding [61, 169]. Since context can provide rich information about why, where and a piece of knowledge is applied, it is very necessary to consider the context during the use of knowledge to improve the applicability of knowledge [46].

Knowledge management has been defined as "the capability by which communities capture the knowledge that is critical to their success, constantly improve it, and make it available in the most effective manner to those who need it" [44]. Context has been considered as a critical concept in knowledge management, where the relevant architectures should include the design of knowledge elements as well as the design of the overall contextual characteristics of the knowledge and the relationships among them[376]. In this situation, knowledge artifacts need to be equipped with context-based features so that they can distribute information effectively within the application domain and relates to other specific knowledge more evenly across the organization[100].

12.3 Design of the Ontology

The basic design concept of our ontology is to build linkages with contextual software scenarios (according to the application context), and the corresponding contextualized security knowledge, in which the critical security concepts are drawn from the security domain model as common vocabularies. (See Figure 12.1).

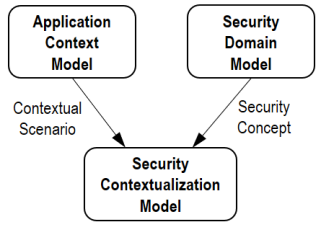


Figure 12.1: Three models span the modeling of contextualized security knowledge

12.3.2 Application Context Modeling

The context model defined a complete representation of what context is in a particular domain. In our ontology, the context for software security knowledge is supported by the creation of scenarios in different application contexts. Contextual scenarios refer to different manifestations within a context [130]. The scenario presents a snapshot of possible features and corresponding code fragments in the specific functionality. For example, regarding the application functionality of "Generating HTML pages" in the web application context there includes a set of scenarios, such as generating static or dynamic pages, and using external data from HTTP requests or data stores. We choose a scenario-based approach because scenarios can be easily adapted to the situation of the represented applications and can be easily integrated with the conceptual security knowledge. It also draws on situated security knowledge, that is, understandings particular to the application context in which they generate. Figure 12.2 represents the application context model used in the ontology. In the context modeling, in addition to scenarios, we focus on characteristics that are highly relevant for retrieval within a software application, concerning three perspectives:

- The functional area (and the corresponding functionalities) that the application is associated with.
- The application category that scenario/functionality belongs to.
- The platforms that the scenario functionality is used.

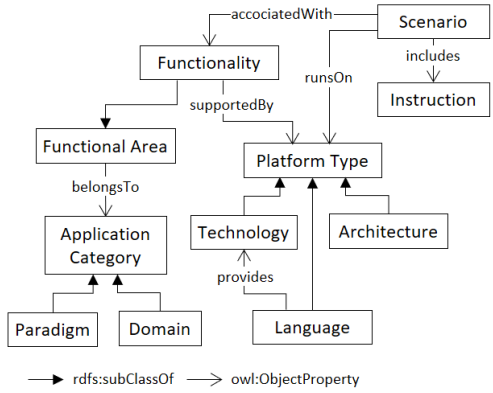


Figure 12.2: Application Context Model

Application category: It is a set of characteristics to categorize software applications, in which two sub-classes are included: *Paradigms* (e.g., web, mobile, and desktop applications, etc.) and *Domains* (e.g., banking, health, and logistics applications, etc.).

Platform type: This superclass specifies programming languages, technologies, and architectures that are used to create the software application. *Technology* can be provided by a certain programming language. For example, Silverlight is the technology that has been implemented in C# language, while J2EE is the subset of

Java technologies. *Architectures* refer to the fundamental system structure to operate the application, such as the MySQL database management system and an Android operating system.

Functional area: It is a group of application functionalities, which represents an aspect of software applications that can be performed by users or other systems in a particular application category. For example, "Outputting HTML" is a functional area in the web applications paradigm, in which "Generating HTML dynamically using user-supplied data" is one of the functionalities. A functionality is supported and run on some combinations of platform types.

12.3.3 Security Domain Modeling

The security domain model describes the knowledge, which is of teaching subjections through a set of concepts. Figure 12.3 illustrates the security concepts and their relationships in the security domain model. In this model, we aim to define the security knowledge schema that is easier to be formed as metal models of learners whiling learning about software security. For this purpose, the schema should be simplified and remain focused on the objective of reducing the content load. In general, our intention is to guide users in answering three questions while dealing with software scenarios:

(1) What are the possible attacks?

(2) Why does the software encounter attacks?

(3) How can these attacks be prevented or mitigated?

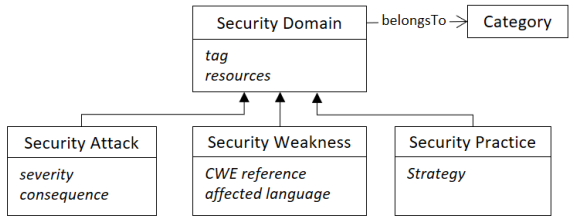


Figure 12.3: Security domain Model

In accordance with such design considerations, we identified three security concepts that are most widely used throughout the security domain and need to be concentrated learning on. Ultimately, three classes were incorporated into the security conceptualization model: Security Attack, Security Weakness, and Security Practice. The definitions of the three security concepts are given in the following:

- *Security Attack*: It represents actions taken against the software application with the intention of doing harm. Examples are SQL injection, Cross-Site Scripting (XSS), etc. Security attacks exploit security weakness existed in software applications.

- *Security Practice*: It represents methods, procedures or techniques to prevent security weakness. Examples are "Input validation" and "Output encoding" in preventing XSS.
- *Security Weakness*: It represents bug, flaws, vulnerabilities and other errors that exist in the software applications. Examples are "Improper to neutralize input during HTML generation" and "Fail to perform a bound check while copying data into memory stack".

From a security conceptualization point of view, we only want to indicate which principles or abstract ideas are needed, not their practical implementation. Therefore, we describe security knowledge in this model at a level of abstraction. The instances of these classes specify only the fundamental characteristics of the security concepts, not specific software application aspects. The major advantage of such design is to enhance the comprehension of the conceptual security knowledge among various security contexts. Furthermore, we adopt an abstract class *Security Domain* as a superclass for all security concepts. In the security conceptualization model, we apply segmentation of interests so that only generic descriptions remain as attributes in the class *Security Domain*. Additionally, we create a *Category* class, in which security concepts can be allowed grouping in categories.

12.3.4 Security Contextualization Modeling

To help users gain a more flexible understanding of the study concept in a range of situations with varying levels of abstraction, we organize security knowledge by blending abstract and concrete perspectives. The term contextualization is used here to describe the process of drawing specific connections between security domain knowledge being taught and an application context in which the abstract knowledge can be relevantly applied or illustrated. In this study, abstract knowledge refers to the conceptual security domain knowledge, while concrete knowledge relates to the contextualized scenario-specific security knowledge. Research has shown that presenting knowledge in both concrete and abstract terms are far more powerful than presenting either one in isolation [348].

To this extent, the security contextualization modeling manages security knowledge in the context of specific scenarios and brings together the conceptual knowledge that is described in the security conceptualization model. The including security concepts are aligned with those defined in the security conceptualization model, which are *Security Attack*, *Security Weakness*, and *Security Practice*. In order to clearly state the purposes and distinguish them from the security conceptualization model, we used different classes, namely *Concrete Security Attack*, *Concrete Security Weakness*, and *Concrete Security Practice*. Figure 12.4 illustrates the security contextualization modeling. The abstract class *Contextualized Knowledge* is used from which these three classes inherit common attributes such as tags or external resources. Once the conceptualization knowledge model is defined, each security concept can be connected to the corresponding classes in the security conceptualization model.

Figure 12.5 depicts the full view of the ontology-based knowledge model, including the interrelationships of the components

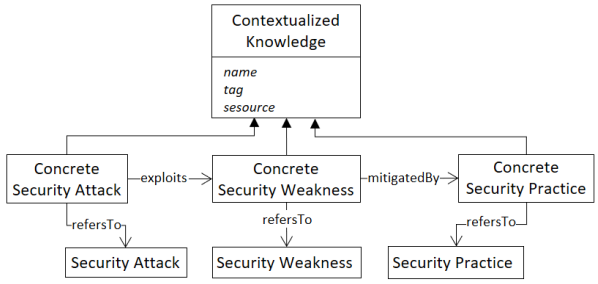


Figure 12.4: Security contextualization model

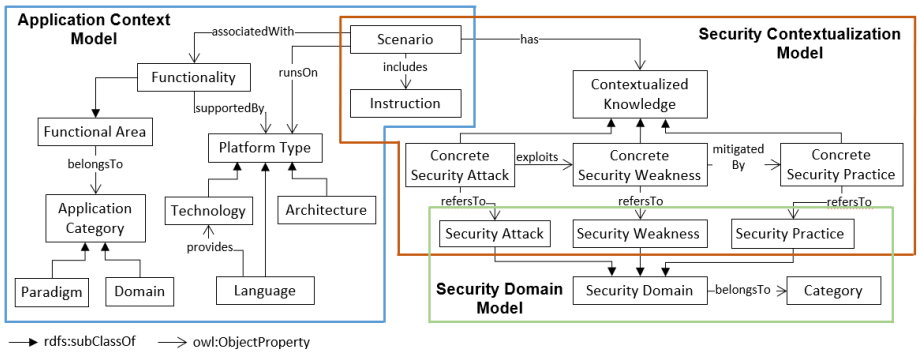


Figure 12.5: The ontology-based security knowledge model

12.4 Evaluation of the Ontology

To validate the effectiveness of the ontology, we conducted several evaluation phases. The overall evaluation process that we undertook is shown in Figure 12.6.

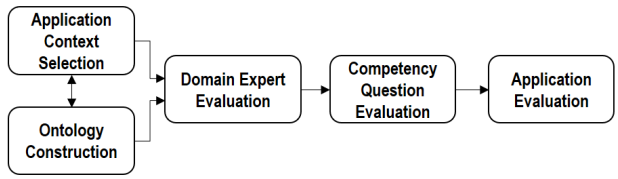


Figure 12.6: The ontology evaluation process

First, in order to evaluate the proposed ontology with a real-world case, we chose a Web Application paradigm with Flat PHP technology as the application context of this pilot study. Functionalities, scenarios and security knowledge items (attacks, weaknesses, and practices) were collected under the defined context. The ontology (concepts and relationships) were implemented used the Protégé tool [20] with Ontology Web Language (OWL, https://www.w3.org/OWL). Figure 12.7 depicts the

ontology design in Protégé editor whereas Figure 12.8 presents the maintenance of object properties and data properties for contextualized knowledge (Security Attack).

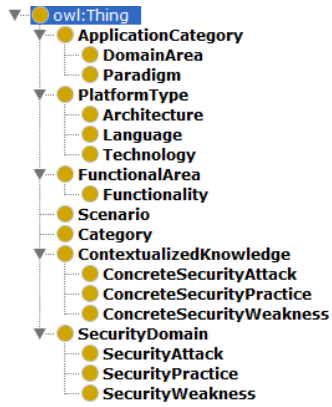


Figure 12.7: Ontology design in Protégé editor

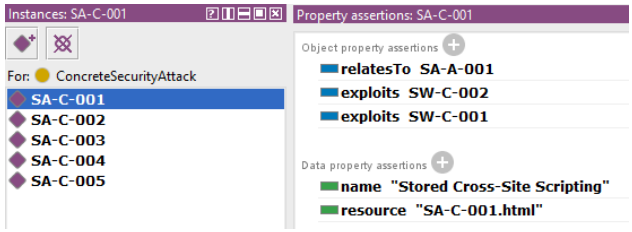


Figure 12.8: The objective property and data property of concrete knowledge (Security Attack)

The domain expert evaluation was carried out by an internal security professional within NTNU who provided the competencies using a computer/cybersecurity, and ontology building method and analysis. The ontology structure, including concept definitions and relations, were reviewed and analyzed. A few weaknesses were identified:

(1) Difficulty to model software technologies and architectures in application context model,

(2) No category classes to group knowledge items in the security domain model, and,

(3) No vulnerability concepts in the security domain model.

We considered comments (1) and (2), and have issued change requests of the ontology design, in which a *Category* class was created in the security domain model, whereas the class of *Platform type* was split into two sub-concepts, namely *Technology* and *Architecture*. In the domain model, we did not differentiate between terms *Security Weakness* and *Vulnerability*, as *Security Weakness* is naturally a more general class that

could cover different security errors, such as design flaw and coding errors. Therefore, the idea in (3), which suggested incorporating concepts of vulnerability, was shelved.

After taking the review comments and mitigate the identified weakness, the ontology was evaluated with competency questions against its initial requirements. Therefore, two exemplary questions were developed:

Q_1: What are the available software scenarios given in the functionality "Generating output in web pages using user-supplied data", PHP language and MySQL database?

Q_2: What are the relevant contextualized knowledge items of the first scenario from the result of the question (a)?

To answer the above competency questions, we used SPARQL protocol [21] to extract information from the RDF graph. Two corresponding SPARQL statements were prepared and executed in Protégé editor. Figure 12.9 demonstrates querying scenarios using the given functionality and platform types (programming language and architecture), from which Q_1 can be answered. For the answer of Q_2, Figure 12.10 shows the query result that returns the instances of contextualized security knowledge of a specific scenario, and the short names of related security domain knowledge.

SPARQL query:

```
PREFIX onto: <http://www.semanticweb.org/ContextOnto.owl#>
SELECT  ?Functionality ?functionality_name ?Language ?Architecture ?Scenario
        WHERE {
                ?Scenario onto:associatedWith ?Functionality.
                ?Scenario onto:runsOn ?Language.
                ?Scenario onto:runsOn ?Architecture.
                ?Functionality onto:functionality_name ?functionality_name.
                FILTER regex(str(?Functionality),'F001', 'i').
                FILTER regex(str(?Language),'PHP', 'i').
                FILTER regex(str(?Architecture),'mySQL', 'i')
}
```

Functionality	functionality_name	Language	Architecture	Scenario
F001	"Generate output in web pages using user supplied data"	PHP	mySQL	S001
F001	"Generate output in web pages using user supplied data"	PHP	mySQL	S003
F001	"Generate output in web pages using user supplied data"	PHP	mySQL	S002

Figure 12.9: An example of SPARQL (to query Scenarios)

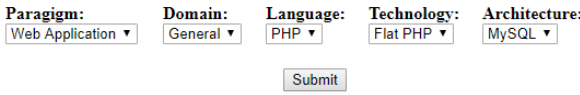
SPARQL query:

```
PREFIX onto: <http://www.semanticweb.org/ContextOnto.owl#>
SELECT ?ConcreteSecurityAttack ?ConcreteSecurityWeakness  ?ConcreteSecurityPractice
       ?attack_name ?weakness_name ?practice_name
            WHERE {
                    ?Scenario onto:has ?ConcreteSecurityAttack.
                    ?ConcreteSecurityAttack onto:exploits ?ConcreteSecurityWeakness.
                    ?ConcreteSecurityWeakness onto:mitigatedBy ?ConcreteSecurityPractice.
                    ?ConcreteSecurityAttack onto:relatesTo ?SecurityAttack.
                    ?SecurityAttack onto:attack_name ?attack_name.
                    ?ConcreteSecurityWeakness onto:relatesTo ?SecurityWeakness.
                    ?SecurityWeakness onto:weakness_name ?weakness_name.
                    ?ConcreteSecurityPractice onto:relatesTo ?SecurityPractice.
                    ?SecurityPractice onto:practice_name ?practice_name.
                    FILTER regex(str(?Scenario),'S001', 'i')
}
```

ConcreteSecurityAttack	ConcreteSecurityWea...	ConcreteSecurityPrac...	attack_name	weakness_name	practice_name
SA-C-001	SW-C-001	SP-C-001	"Stored Cross-Site Scripting"	"Improper Neutralization of input	"Input validation"@
SA-C-001	SW-C-002	SP-C-002	"Stored Cross-Site Scripting"	"Failure to escape/encode ouput'	"Output encoding"@

Figure 12.10: An example of SPARQL (to query security knowledge)

After the domain expert's review with a competency-question examination, we took a further application-based evaluation approach [56, 205] by plugging the ontology into an application for further evaluation. For this purpose, we developed a web-based application prototype based on our proposed ontology. The objective of this application is to present scenario-based security knowledge that is both concrete and abstract, according to the contextual information that the user provides. The front-end was designed as a web-based user interface with HTML and JavaScript languages. The backend was implemented with Java, and the ontology repository was accessed with Jena API (https://jena.apache.org), which is a Java framework using for building semantic web applications. Jena has the advantage that it provides wild-ranging Java libraries to help developers handle OWL, and SPARQL conformed with W3C recommendations. Figure 12.11 presents the user interface of the context menu, in which the learner selects relevant criteria based on the desired knowledge (or prior programming experience) to scope the functionalities and corresponding scenarios.



Functionality	Scenario
Generate output in web pages using user supplied data	S001 S002 S003
Accessing database using user input	S006
Performing a site function using a static URL or POST request	S011 S012

Figure 12.11: The user interface for context selection

In presenting the security knowledge, the web page is mainly composed of four framesets: the security knowledge structure, the scenario description, the contextualized knowledge, and conceptual knowledge. Figure 12.12 shows a snapshot of the knowledge presentation. The scenario here is used as a starting point to browse security knowledge, which is made up of practical demonstrations of the pre-described application functionality and the code fragments that were extracted from the class Scenario and Instruction. To present practical security knowledge for the scenario, which is contextualized knowledge, we extracted information from classes under the Contextualized Knowledge superclass, which includes perceptually detailed and rich materials from ontology, such as security attacks with different exploits, coding mistakes, and the corresponding secure coding practices. In addition to the practical knowledge, users can also capture abstract explanation as well, that is conceptual knowledge from Security Domain classes.

12.5 Discussion

Ontology technologies have become the core component of today's applications such as electronic commerce, knowledge portals, information integration and sharing, and web services [180, 331, 446, 474]. Our ontology approached the role of ontologies in

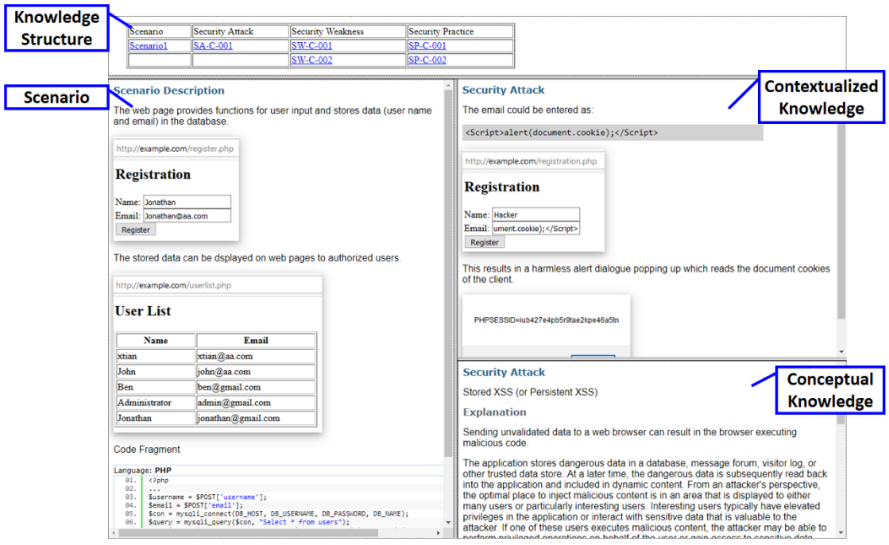


Figure 12.12: The user interface for security knowledge presentation

managing contextualized knowledge in the domain of software security. With the context-based design approach, a dynamic situational application scenario can be integrated

together with the conceptual security domain knowledge. The advantage of this ontology model is twofold. First, it separates concrete and abstract security knowledge in two models, which simplified knowledge maintenance and retrieval. Second, it shares a common understanding of security concepts between security domain and contextualization models to enable semantic interoperability.

In addition to the application scenario demonstrated in the previous section, this ontology can be also used in various settings. For example, in the pedagogical environment, a course tutor, who is engaged in the introduction of security vulnerabilities, can use the proposed ontology to quickly identify a number of real-world examples of facing a specific security attack or vulnerability, to improve the effectiveness of learning. In the practical software development process, software engineers are allowed to find solutions to exceptional situations by searching for similar contexts. For example, a PHP web application designer can refer to another security setup by looking for a similar domain and software technologies. The presence of detailed information on the relation between classes can enable answering the various questions related to security tasks. Furthermore, since our ontology is developed using the OWL standard in the Protégé tool, it enables the possibility to be used by an automated tool to provide advanced services such as more accurate security requirements and design suggestions.

Yet, the software security domain is complex and dynamic. New threats and countermeasures are continuously evolving. Although the approach described here provides technologies to store and present security knowledge, security experts or practitioners' involvement is crucial to fill the security ontologies with the necessary information and then to apply them in security education and the practical software development process.

12.6 Related Work

There have been extensive research works in the area of security knowledge modeling and ontology applications to software security. Some papers focus on using an ontology to model security vulnerabilities. Guo and Wang [183] presented an ontology-based approach to model security vulnerabilities listed in CVE (Common Vulnerability and Exposure, https://cve.mitre.org/). The authors identified critical concepts of security vulnerabilities in the domain of software security, which can be provided for machine-understandable CVE vulnerability knowledge and reusable security vulnerabilities interoperability. Syed and Zhong proposed an ontology-based conceptual model for the formal knowledge representation of the cybersecurity vulnerability domain and intelligence, which integrated cybersecurity vulnerability concepts from several sources including CVE, NVD (National Vulnerability Database, https://nvd.nist.gov/), CVSS (Common Vulnerability Scoring System, https://www.first.org/cvss/) framework, and social media. Alqahtani et al. [14] proposed an ontological representation, which establishes links with bi-directional traceability between traditional software repositories (e.g., issue trackers, version control systems, Q&A repositories) and security vulnerabilities databases (e.g., NVD)

Some researchers presented their ontology in supporting security requirements and design processes in software development. Gyrard et al. [185] proposed STACK ontology (Security Toolbox: Attacks & Countermeasures) that supported developers in secure application design. Countermeasures in STACK included cryptographic concepts (encryption algorithm, key management, digital signature, and hash function), security tools, and security protocols. Kang and Liang [227] presented the security ontology adopting the Model Driven Architecture (MDA) methodology. Their proposed ontology could be used in security concepts modeling in each phase of the development process (e.g., the requirement and design phases) with MDA. In order to improve the application of security patterns to the security engineering domain, Guan et al. [182] proposed an ontological approach facilitating security knowledge mapping from security requirements to security patterns. Manzoor et al. [282] developed an ontology, illustrating the relationships across various actors involved in the Cloud ecosystem, to analyze different threats to/from Cloud-system actors.

Finally, some efforts focused on building security ontology specifically in the context of web application development. Salini and Kanmani [388] presented an ontology for defining the security requirements of web applications. The included concepts are

assets, vulnerabilities, threats, and stakeholders. Their ontology aimed at reusing the knowledge of security requirements in the development of different kinds of web applications. Buch and Wirsing [65] presented a security ontology for secure web applications (SecWAO), which aimed to support web developers to specify security requirements or make design decisions in web application development. It distinguished various concepts among methods, tools, mechanisms, assets, vulnerabilities, and threats. Velasco et al. [455] presented an ontology-based framework for string, presenting and reusing security requirements. Their framework integrated security standards, methods of risk analysis, and the requirements ontology.

A major feature, which is common for all the above studies, is that the ontology is security driven, focusing on unifying security concepts and terminology. Subsequently, they either dedicate to a certain software domain or support part(s) software development processes. Our ontology approach differentiates from the previous research work in the following aspects:

(1) Our ontology is context-based, which models security knowledge with a diversity of software features and technologies;

(2) Our ontology describes security knowledge with a contextual situation, and meanwhile, complements the concrete knowledge with abstract description.

12.7 Conclusion and Future Work

This paper presents a novel approach for modeling software-security knowledge with a context-based approach, in which the security knowledge can be retrieved taking the context of the software application into consideration. The design of our ontology ensures that users understand the security-relevant aspects of critical software features. In addition, software developers are able to identify the possible attacks and security errors efficiently that are associated with the functionalities of their software products, based on the domain of the application, the programming language or technologies they used, In this paper, we have presented the core concepts of the ontology, as well as an evaluation with an application scenario. Our proposal is deemed useful for security researchers who wish to formalize and manage contextualized knowledge in their domain, systems, and methods.

In future work, we expect to expand the ontology continuously, enriching the knowledge content by including more software scenarios with a broad application context, while also providing contextual details in branches of security domain knowledge and enriching the abstract explanations. We also plan to have further evaluation of the modeling approach with educators and security experts in the domain of information security. We believe that such a context-based approach in ontology modeling can benefit border security domains, such as network security and cryptography. Furthermore, we intend to enhance and complete a learning system

for software security based on this ontology. The ultimate goal of our research is to create conditions for more effective learning about software security, which can motivate learners and stimulate their interest.

Chapter 13

Development of Ontology-Based Software Security Learning System with Contextualized Learning Approach

Wen, Shao-Fang and Katt, Basel. "Development of Ontology-Based Software Security Learning System with Contextualized Learning Approaches." Journal of Advances in Information Technology. 2019, volume 10, no. 3, pp 81-90

Author Contributions— Initial conceptualization the research and the prototyped system were developed by Shao-Fang Wen. The research methodology and system evaluation design and were reviewed by Basel Katt.

Abstract— Learning software security is one of the most challenging tasks in the information technology sector due to the vast amount of security knowledge and the difficulties in understanding the practical applications. The traditional teaching and learning materials, which are usually organized topically and security-centric, have fewer linkages with learners' experience and prior knowledge. Learners often do not associate vulnerabilities or coding practices with programs similar to what they were writing in their previous time. Consequently, their motivation for learning is not touched by conventional methods. In this paper, we present a software-security learning system based on ontologies that facilitates the contextual learning process by providing contextualized access to security knowledge via real software application scenarios, in which learners can explore and relate the security knowledge to the context they are already familiar with.

13.1 Introduction

Software security has been a subject of a plethora of studies for at least 40 years, and a steady stream of innovations has improved software engineers' ability to secure software development and to protect applications. Improving software security requires many different approaches. One way is to give software engineers or learners the knowledge and skills to resist attacks and handle errors appropriately [46]. To emphasize security, a relatively large number of best practices and vulnerability information have been published by security committees in publications or on the internet. To this extent, the huge amount of information has resulted in a form of information overload for learners. Moreover, the domain of software security is quite context-specific and can be applied in diverse ways [294]. As a result, learning software security becomes a complex and difficult task because learners must not only deal with a vast amount of knowledge about a variety of concepts and methods but also need to demonstrate the applicability of the knowledge through experience in order to understand their practical use.

In traditional software security teaching, little attention is given to what a real-world situation really means to learners, and there is not much content addressing the connection between the security concepts and learner' prior knowledge. In conventional security learning materials, the knowledge content is commonly security-centric and organized topically, which distinguishes two fundamental segments: the white-hat approach, where the main emphasis on security principles and anti-attack mechanisms, and the black-hat, which teaches how to break software and how malicious hackers write exploits. These learning materials are often described in the form of a reference manual or a guide to particular security subjects. The topical knowledge organization is useful for rote memorization of a specific security subject or for information reference later; however, it is difficult for learners to understand the rationale of the topics, and correlate those topics with real software scenarios. Learners usually finish reading such materials with little understanding of the context in which the security knowledge should be applied, or with the feeling that the security domain is so extensive and software, security is so difficult to achieve that they simply cast it aside.

We argue that the way learners process security information and their motivation for learning are not touched by conventional methods. Research indicates that learning is most efficient when it is linked with experience and prior knowledge that students bring to a given learning situation [90, 266]. However, novice learners do not always make connections between new information and prior knowledge or everyday experiences in ways that are productive for learning [259]. In the context of software security learning, learners interpret security knowledge they gain with a range of strongly held personal programming experience. They often do not associate vulnerabilities with programs similar to what they were writing in their previous time. As the suggestion given in the research of engineering education [137], establishing the relevance of learning materials before going into the details can

provide the concrete experience that starts the learning process. In order to regulate learning about software security effectively, security knowledge should be contextualized in a meaningful scenario where they can learn security principles and processes with a real-world situation.

Our primary objective is to create conditions for more effective learning for software security that can motivate learners and stimulate their interests. This paper is part of an investigation into contextualized learning in the domain of software security. We propose a learning system, which facilitates the contextual learning process by providing contextualized access to security knowledge through real software application scenarios. This learning system is a place where learners can explore and relate the security knowledge to the context they are already familiar with. To develop this kind of learning system, the security knowledge should be modeled and managed in a manner where the knowledge can be retrieved taking the context of the application in hand into consideration. Ontologies make it possible to give this kind of purpose since it facilitates the capture and construction of domain knowledge and enables the representation of skeletal knowledge to facilitate the integration of knowledge bases irrespective of the heterogeneity of knowledge sources [181]. This paper presents the proposed design approach of the contextualized learning system and the developed proof-of-concept prototype.

The rest of this paper is organized as follows: In section 13.2, we introduce the theoretical background of this study. Section 13.3 reviews the related work on ontology approaches in the software security domain. In section 13.4, we describe the design approach of the contextualized learning system. Section 13.5 presents the detailed design of the underlying ontology of the learning system. Section 13.6 describes the developed prototype using the proposed approach. Lastly, the conclusion and future works are presented in section13.7.

13.2 Theoretical Background

The theoretical background of this research is drawn from the field of context-based knowledge and contextualized learning. According to Anind K. Dey [117], context is ''A set of information used to characterize a situation of an entity". Nonaka [328] indicates that knowledge reflects a particular stance, perspective, or intention in accordance with the characteristics of a specific context, which is different from information. According to Brézillon [58, 61], knowledge comes from a variety of context and it cannot be accurately understood without context. Without proper contextual information, knowledge can be isolated from other relevant knowledge resulting in limited or distorted understanding [169]. Researchers of psychology and education indicate when knowledge is learned in a context similar to that in which the skills will actually be needed, the application of learning to the new context may be more likely [117, 122, 352]. Predmore [360] shows that learning about knowledge content within real-world experience is important because "once [students] can see the real-world relevance of what they're learning, they become interested and

motivated". Since context can give guidance about when, where and why a piece of knowledge is used, considering the context in knowledge use is very necessary to enhance the applicability of knowledge [46].

Contextualized Teaching and learning builds upon a similar concept of putting learning activities into perspective to achieve the best teaching and learning outcomes. Researchers Berns and Erickson define contextualized learning as a practice that endeavors to link theoretical constructs that are taught during learning, to a practical, real-world context. The underlying theme behind contextual learning activities is simple. It recognizes that by embedding instructions in contexts that adult learners are familiar with, learners more readily understand and assimilate those instructions. Contextualized instruction in general, starts with presenting a context from which the concepts are developed on a need-to-know basis. This requires teachers to teach in a more constructivist way, i.e. to position the concepts of the learning subject in contexts recognizable to students and to stimulate the active learning of the students [346]. The contextualization of the learning on demand can not only be seen from the point of view of an actual problem or learning situation but also in a longer-lasting process of learning activities that are integrated [425].

In computer science education, there is also a broad agreement that teaching units should start from a "real-world" context or phenomenon, aiming to create connections to prior knowledge, to increase the relevance of the material to students or to show application situations of the intended knowledge, thereby increasing motivation [120, 184]. These contrast with more traditional approaches that cover abstract ideas first, before looking at practical applications. Likewise, in software engineering, studying from a context and then abstracting the knowledge gained to be able to use it in a new context is a common way of learning programming that has been observed extensively in both new and experienced programmers [23, 243]. In order to capture and use security knowledge appropriately, it is necessary to first specify which context information is to be handled, and then represent this in a format that is understandable and acceptable to the individuals. Thus, a context for a software security topic includes the circumstances in which its technical content exists. Therefore, to talk about software security in context is to say that knowledge would not only include the basic principles and processes of software security but would consider how security knowledge is used in one or more particular domains or application areas.

13.3 Related Work

In this section, we describe research works related to this study from the viewpoint of knowledge modeling support for software security based on ontology. According to Gruber [180], an ontology is "an explicit and formal specification of a conceptualization", that is, a formal description of the relevant concepts and relationships in an area of interest, simplifying and abstracting the view of the world

for some purpose [473]. There have been a number of papers published in the area of ontology modeling and applying semantic technologies to software security. Some efforts focused on building security ontology to model the security requirements. Salini and Kanmani [388] present an ontology of security requirements for web applications, including concepts of assets, vulnerabilities, threats, and stakeholders. Their work aims at enabling the reuse of knowledge about security requirements in the development of different web applications. Buch and Wirsing [65] present the SecWAO ontology with a focus on a secure web application, which aims to support web developers when specifying security requirements or making design decisions. It distinguishes concepts (classes) between methods, notations, tools, categories, assets, security properties, vulnerabilities, and threats.

Some research works to present their ontology to support security design and risk assessment. Gyrard et al. [185] present the STACK ontology (Security Toolbox: Attacks & Countermeasures) to aid developers in the design of secure applications. STACK defines security concepts such as attacks, countermeasures, security properties, and their relationships. Countermeasures can be cryptographic concepts (encryption algorithm, key management, digital signature, and hash function), security tools, or security protocols. Kang and Liang [227] present a security ontology with the Model Driven Architecture (MDA) approach for the use in the software development process. The proposed ontology shows that the proposed security ontology can be used in modeling and designing security issues and concepts in each phase of the development process with MDA. Marques and Ralha [287] propose an ontology, which is related to the risk management aspect of web-based system development. The model is mainly employed in the design phase of the system development.

Finally, there are some papers focusing on using an ontology to model vulnerabilities and security attacks. Guo and Wang [183] present an ontology-based approach to model security vulnerabilities listed in Common Vulnerabilities and Exposures (CVE). The authors captured important concepts for describing vulnerabilities in the context of software security, providing machine-understandable CVE vulnerability knowledge and reusable security vulnerabilities interoperability. Khairkar et al. [232] present an ontology to detect attacks on web systems. The authors use semantic web concepts and ontologies to analyze security logs to identify potential security issues. This work aims to extract semantic relationships between attacks and intrusions in an Intrusion Detection System (IDS). Razzaq et al. [369] propose an ontology of attacks and an ontology of communication protocols, which provide a construct to improve the detection capability of application-level attacks in web application security. The authors employ the use of semantics in application layer security contrary to tradition signature-based approaches.

13.4 Design Approach

To facilitate contextualized learning about software security and create engaging learning experiences for learners, we proposed a contextualized approach for software-security learning with three strategies: (1) Starting with a meaningful scenario; (2) Stimulating learners' mental model for software security learning; and (3) Moving from concrete to abstract security knowledge. Figure 13.1 depicts an abstract representation of our design approach to the learning system for software security. Learners will engage in the learning process by taking advantage of relevant knowledge content. We describe in detail these strategies in the following sections.

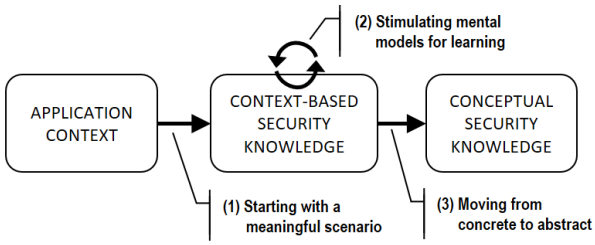


Figure 13.1: The design approach of the learning system

13.4.2 Starting with a Meaningful Scenario

Contextualized learning often takes the form of real-world examples of problems that are meaningful to the learners personally [373]. Creating the relevance of the learning knowledge before going into the details could provide a stronger foundation for the learning process. Therefore, to begin the process of learning, a meaningful situation for learners must first be established. In our study, the learning situations are created through the use of contextual scenarios in the application context, which utilize some form of anchoring situation events [85] to engage learners with security concepts that are addressed in the software problem or situation. Contextual scenarios refer to different manifestations within a context [130]. We choose a scenario-based approach because scenarios can be easily adapted to the situation of the represented applications and can be easily integrated with the conceptual security knowledge.

An anchoring event (i.e., the scenario in our study), enabling learners to visualize how the knowledge substance relates to their prior experience [85], could be revisited repeatedly during the learning sessions. For instance, regarding the application functionality of "Generating HTML pages" in the web application context there includes a set of scenarios, such as generating static or dynamic pages, and using external data from HTTP requests or data stores. Those scenarios can serve as anchoring events to evoke the learners' memories of programming and draw attention to software events and conditions. Research has shown that using anchoring events in learning promotes memory recall and the subsequent transfer of information to a new setting [85], meanwhile, helps render abstract ideas more

concretely and thus provides a cognitive mooring around which newly learned ideas can be linked with learners' prior understandings [86]. The use of anchor evens in our study aims to echo learners' real-world experiences to context-based security knowledge to help learners apply their emerging understandings about software security to the real software cases, thus helping them see value in their learning sessions.

13.4.3 Stimulating Mental Models for Learning

Contextual learning is a learning approach that ties brain actions in creating patterns that have meaning [113]. In order to help learners make sense of complex security knowledge and create a strong and lasting bond among security concepts while they are engaged through various anchoring events, our strategy is to elicit learners' mental models for the navigation of security knowledge. Kenneth Craik [94] suggested that the human mind builds and constructs "small-scale models" to anticipate events. Such mental models allow learners to gain insight regarding their world by building a work scheme [160], which makes it easier for them to access the information needed to understand the knowledge domain, make predictions, and decide upon action to take [379]. This can result in successful learning by engaging students, fostering their concentration, and assisting them in organizing systemic information [402].

Mental models combine a schema or a knowledge structure with a process for manipulating the information in the memory [304], where the knowledge structure interrelates a collection of facts or concepts about a particular topic [494]. In order to be useful explanatorily, a mental model has to have a similar relation-structure to the reality it models. Then the constructed mental model can be used to answer questions or solve problems [235]. Generally, our intention was to guide learners in answering three questions while dealing with each anchoring event:

- What are the possible attacks?
- Why does it encounter attacks?
- How can these attacks be prevented?

The knowledge structure serves as the basis for both knowledge retention and retrieval, as well as transfer. Once learners answer what–why–how questions, the relationships between the security concepts are revealed in their midst, and thus, their representation of mental models expands.

13.4.4 Moving from Concrete to Abstract Knowledge

To help learners gain a more flexible understanding of the study concept in a range of situations with varying levels of abstraction, we organize security knowledge by blending abstract and concrete perspectives; presenting it with a sequence from concrete to abstract. In our study, abstract knowledge refers to the conceptual security

domain knowledge while concrete knowledge relates to the contextualized scenario-specific security knowledge. Research has shown that presenting knowledge in both concrete and abstract terms are far more powerful than presenting either one in isolation [348]. Lave and Wenger [264] also argued that abstract and generalized knowledge gains its power through the expert's ability to apply it in specific situations. The used concrete-to-abstract approach in knowledge presentation differs from the traditional, where the concepts are of foremost importance and are usually explained first before concrete examples and applications are discussed. Consequently, learners may struggle to finish reading them due to a learning style mismatch. Several studies [136, 291, 292] have shown that the majority of engineering students are sensor-type learners, who like facts, data, and observable phenomena as opposed to theoretical abstractions. Deductive reasoning is facilitated when the domain is familiar and concrete rather than abstract [476].

In such a concrete-to-abstract knowledge presentation, learners discover meaningful relationships between practical functions and abstract knowledge in the context of real applications. The value of concrete representations has been frequently noted in education. Concrete materials can support abstract reasoning because they can be explicitly designed to promote true inferences from perceptual representations to abstract principles [35]. A method known as concreteness fading [170] has the advantage of initially presenting concepts in a concrete fashion and then, over time, augmenting that initial presentation with progressively more abstract representations of the concepts. Abstract understanding is most effectively achieved through experience with perceptually rich, concrete representations [171], while concrete materials make concepts real and therefore easily internalized [226]. As long as the concrete knowledge and the underlying abstract explanation are understood by learners, learning transfers from one context to another will be more effective.

13.5 Underlying Ontology-Based Knowledge Model

One of the central ideas embedded within the learning system is to develop a kernel ontology-based security knowledge model. With this model, the learning application can handle contextualized security knowledge with multiple scenarios in different application-specific contexts and integrates security concepts of security domain knowledge.

13.5.1 Application Context Modeling

The context model represents a definition of what context is in a specific domain. In our ontology, the context for software security knowledge is supported by the creation of scenarios in different application contexts. The scenario presents a snapshot of possible features and corresponding code fragments in the specific functionality that is included in the Instruction class. It also draws on situated security knowledge, that is, understandings particular to the application context in which they

generate. Figure 13.2 represents the application context model used in the ontology. In the context modeling, in addition to scenarios, we focus on characteristics that are highly relevant for retrieval within a software application, concerning three perspectives:

- The application category that scenario/functionality belongs to,
- The platforms that the scenario functionality used, and
- The functional area (and the corresponding functionalities) that the application associated with.

Application category: It is a set of characteristics to categorize software applications, which include two sub-classes: paradigms (e.g., web, mobile, and desktop applications, etc.) and the domains (e.g., banking, health, and logistics applications, etc.).

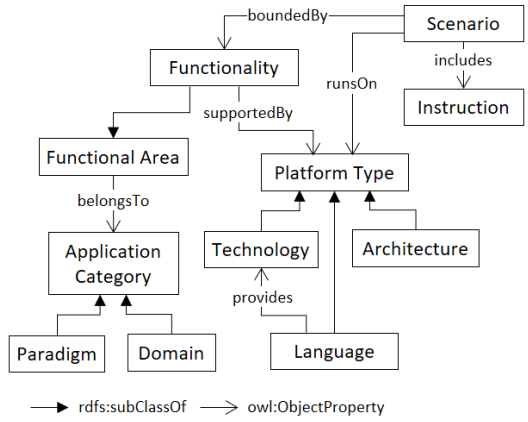


Figure 13.2: Application context model

Platform type: This superclass specifies programming languages, technologies, and architectures that are used to create the software application. Technology can be provided by a certain programming language. For example, Silverlight is the technology that has been implemented in C# language, while J2EE is the subset of Java technologies. Architectures refer to the fundamental system structure to operate the application, such as the MySQL database management system and the Android operating system.

Functional area: It is a group of application functionalities, which represents an aspect of software applications that can be performed by users or other systems in a particular application category. For example, outputting HTML is a functional area in the web-application paradigm, in which generating HTML dynamically using user-supplied data is one of the functionalities. A functionality is supported and run on some combinations of platform types.

13.5.2 Security Domain Modeling

The security domain model describes the knowledge that is an object of teaching through a set of concepts (topics to be taught). In this model, we aim to design a security knowledge structure (schema) that is easier to store in the learners' memory for learning. For the purpose, the schema should be simplified and kept to the point for reducing the content load. We, therefore, identified three security concepts that are most widely used throughout the security domain and need to be concentrated learning on. Ultimately, three classes were incorporated into the security domain model: *Security Attack*, *Security Weakness*, and *Security Practice*. The definitions of the three security concepts are given in the following

Security Attack: It represents actions taken against the software application with the intention of doing harm. Examples are SQL injection, Cross-Site Scripting, etc. Security attacks exploit security weakness existed in software applications.

Security Practice: It represents methods, procedures or techniques to prevent security weakness.

Security Weakness: It represents bug, flaws, vulnerabilities and other errors that exist in the software applications.

From a security conceptualization point of view, we only want to indicate which principles or abstract ideas are needed, not their practical implementation. Therefore, we describe security knowledge in this model at a level of abstraction. The instances of these classes specify only the fundamental characteristics of the security concepts, not specific software application aspects. The main advantage of this design is to share a common understanding of the conceptual security knowledge among different security contexts. Furthermore, we adopt an abstract class *Security Domain* as a superclass for all security concepts. In the security domain model, we apply separation of concerns so that only very general descriptions remain as attributes in the class *Security Domain*. Additionally, for convenience, we allow grouping domain knowledge in categories, which themselves can belong to security concepts. Figure 13.3 illustrates the security concepts and their relationships in the security domain model.

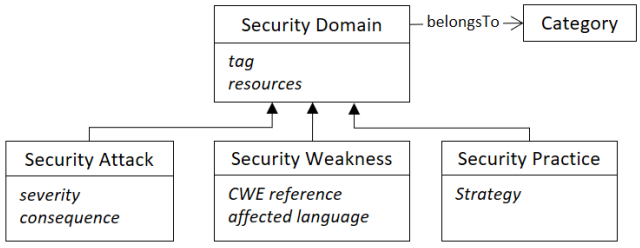


Figure 13.3: Security domain model

13.5.3 Security Contextualization Modeling

Figure 13.4 illustrates the security contextualization modeling. The term contextualization is used here to describe the process of drawing specific connections between security domain knowledge being taught and an application context in which the conceptual knowledge can be relevantly applied or illustrated. To this extent, the security contextualization modeling manages security knowledge in the context of specific scenarios and brings together the conceptual knowledge that is described in the security domain model. The including security concepts are aligned with those defined in the security domain model, which are *Security Attack*, *Security Weakness*, and *Security Practice*. However, in order to clearly state the purposes and distinguish them from the security domain model, we use different classes, namely *Concrete Security Attack*, *Concrete Security Weakness*, and *Concrete Security Practice*. The abstract class *Contextualized Knowledge* is used from which these three classes inherit common attributes such as tags or external resources. Once the conceptualization knowledge model is defined, each security concept is able to be connected to the corresponding classes in the security domain model. Figure 13.5 shows the completed ontology-based knowledge model including the interrelationships of the components.

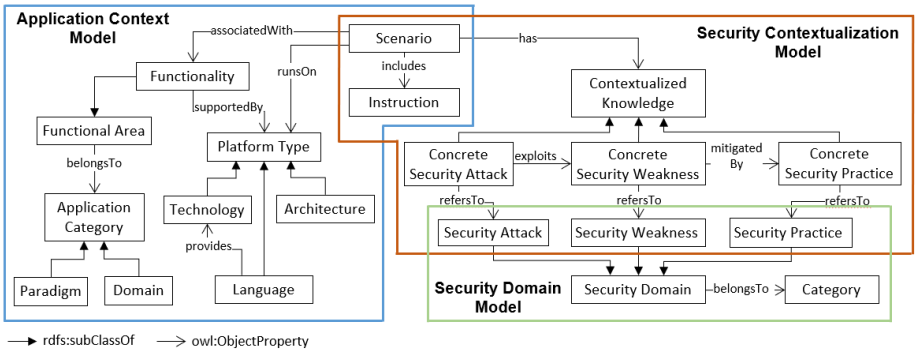


Figure 13.5: The ontology-based security knowledge model

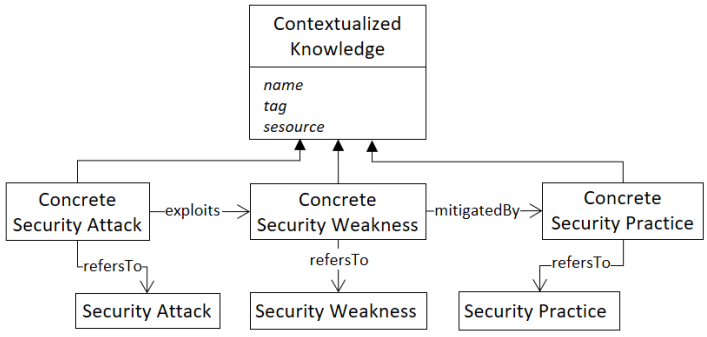


Figure 13.4: Security contextualization model

13.6 The Developed Prototype

We have developed a proof-of-concept prototype to demonstrate the proposed design approach. The high-level system architecture diagram is presented in Figure 13.6. The front-end was designed as a web-based user interface with PHP and JavaScript languages and through it, learners can access the knowledge content. The backend was implemented in Java and access to the ontology repository was provided through the Jena API[39], a Java framework for building semantic web applications. Jena provides extensive Java libraries for helping developers develop code that handles RDF, OWL, and SPARQL in line with published W3C recommendations[40].

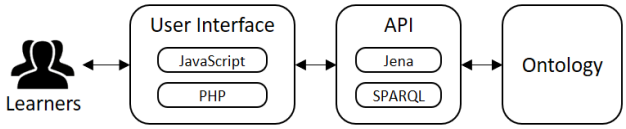


Figure 13.6: High-level system architecture diagram

13.6.2 Construction of the Ontology

To construct the ontology, we used Protégé and OWL Editor because of its simplicity and popularity [444]. When searching the ontology, we use SPARQL protocol to extract information from the RDF graph. Figure 13.7 depicts the ontology design in Protégé editor. An example of SPARQL and the executed result is presented in Figure 13.88. The objective of this query is to return the instances of contextualized security knowledge of a specific scenario, and the short names of related security domain knowledge.

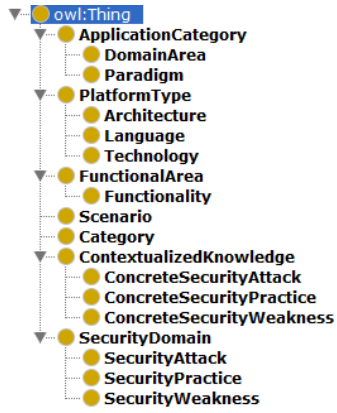
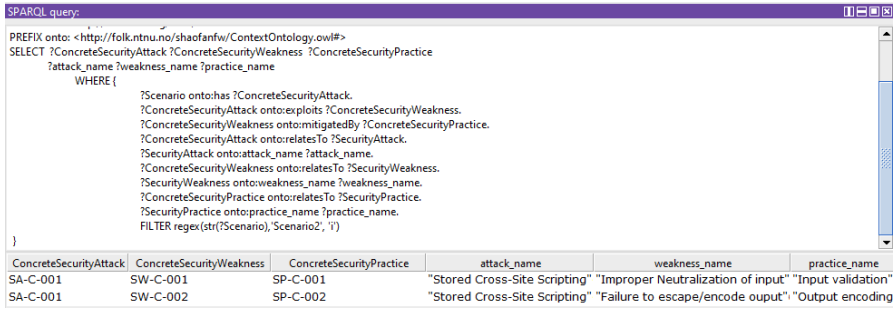


Figure 13.7: Ontology design in Protégé editor

[39] https://jena.apache.org/
[40] https://www.w3.org/2001/sw/


SPARQL query:

```
PREFIX onto: <http://folk.ntnu.no/shaofanfw/ContextOntology.owl#>
SELECT ?ConcreteSecurityAttack ?ConcreteSecurityWeakness ?ConcreteSecurityPractice
        ?attack_name ?weakness_name ?practice_name
            WHERE {
                    ?Scenario onto:has ?ConcreteSecurityAttack.
                    ?ConcreteSecurityAttack onto:exploits ?ConcreteSecurityWeakness.
                    ?ConcreteSecurityWeakness onto:mitigatedBy ?ConcreteSecurityPractice.
                    ?ConcreteSecurityAttack onto:relatesTo ?SecurityAttack.
                    ?SecurityAttack onto:attack_name ?attack_name.
                    ?ConcreteSecurityWeakness onto:relatesTo ?SecurityWeakness.
                    ?SecurityWeakness onto:weakness_name ?weakness_name.
                    ?ConcreteSecurityPractice onto:relatesTo ?SecurityPractice.
                    ?SecurityPractice onto:practice_name ?practice_name.
                    FILTER regex(str(?Scenario),'Scenario2', 'i')
}
```

ConcreteSecurityAttack	ConcreteSecurityWeakness	ConcreteSecurityPractice	attack_name	weakness_name	practice_name
SA-C-001	SW-C-001	SP-C-001	"Stored Cross-Site Scripting"	"Improper Neutralization of input"	"Input validation"
SA-C-001	SW-C-002	SP-C-002	"Stored Cross-Site Scripting"	"Failure to escape/encode ouput"	"Output encoding

Figure 13.8: An example of SPARQL and the executed result

13.6.3 The Process of Learning

The user interface of the prototyped system is presented in Figure 13.9, in which a scenario of HTML output under the web application paradigm is demonstrated. In this prototype, the learning process begins with the concrete in a context familiar to learners and then gradually leads to an understanding of the abstract. Figure 13.10 depicts the learning process that is constructed by the proposed learning system. First of all, a meaningful situation for learners must first be established. The access to learning content in the learning application mainly happens scenario-oriented. We use the scenario as the starting point for learning security concepts on a need-to-know basis while presenting the modeled security knowledge. Based on the desired knowledge the learner selects relevant criteria from the application-context menu to scope the learning scenario. The instructional part of the scenario is made up of practical demonstrations of the pre-described application functionality and the code fragments behind it that bridge the corresponding security knowledge. As described previously, the selected scenario served as an anchoring event that can be view throughout the learning session to anchor learning in the learners' personal experience.

To guide learners navigating through the contextualized knowledge efficiently, it is necessary to illustrate the relationship between the security concepts. On the one hand, it must be transparent for learners about, which causes and effects relevant to the learning content he (or she) is studying. On the other hand, this is essential for learners in order to integrate the semantical impact of the knowledge structure into the mental models for efficient learning. For the purpose, we outline the learning contents in a graphical *Concept Map*, which is shown in the left corner of the system appearance. Concept Map is a visual representation of different concepts and their relationships. Concept mapping help in organizing learners' knowledge by integrating information into a progressively more complex conceptual framework. With the use of concept mapping, the learning arena can be virtualized in a learner's mind [405]. From the visual description, learners extract propositions and create a

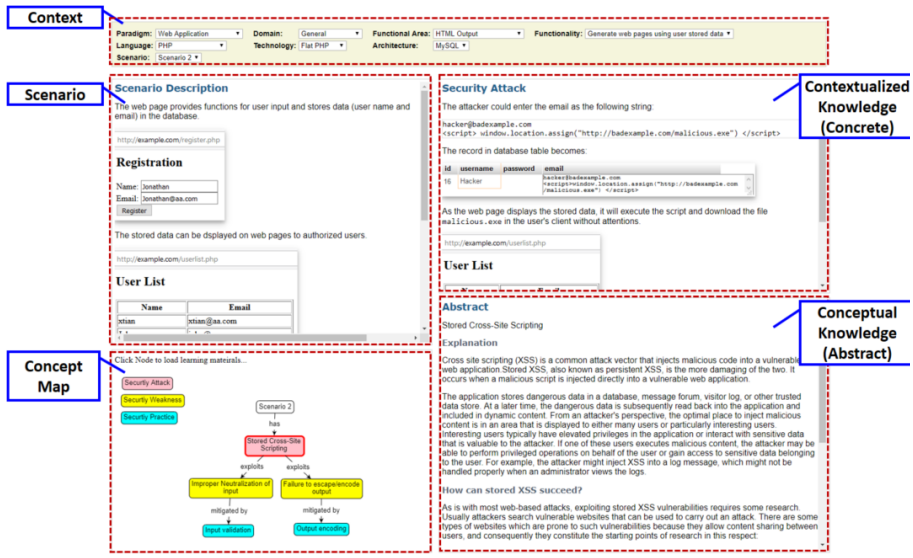


Figure 13.9: The user interface of the developed prototype

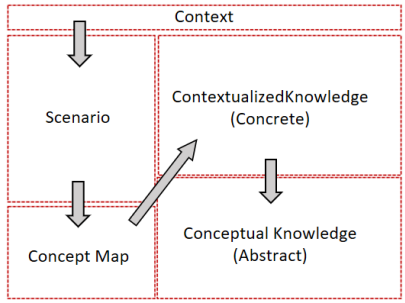


Figure 13.10: The constructed learning process of the learning system

mental model from the graph. Meanwhile, the extracted mental model will be inherently influenced by connecting to their prior experience.

The design of our ontology is able to provide the basis for the development of the concept map of the relationship between these concepts. While a node is clicked on the concept map, the relevant knowledge content is displayed in the right half of the appearance, where the upper part is the contextualized knowledge and the lower part is an abstract explanation, following the concrete-to-abstract presentation strategy. By concrete representations, we include perceptually detailed and rich materials, such as demonstrating security attacks with different exploits, identifying mistakes in the source code, and showing the secure coding practices to fix the mistakes. Figure 13.11 shows a system appearance of viewing the security weakness of the scenario. With scenario-description presenting aside, learners can easily recall features of the context (e.g. code fragment) without interrupting the learning process. After experiencing the

facts, learners then move on to conceptual knowledge, where the abstract explanation is presented. Therefore, dynamic, e.g., situational application scenario is integrated together with the conceptual security domain knowledge. Figure 13.12 presents another scenario in the paradigm of "General implementation" and the language of C/C++. This demonstrated scenario introduces security knowledge related to the functionality of "Performing memory buffer operations using user-supplied data".

13.7 Conclusion and Future Work

This paper presents an ontology-based learning system for software security learning with a contextualized learning approach, which contains three strategies. The first is to establish meaningful scenarios to create a meaningful situation for learners. The design of the application context aims to activate the learner's prior knowledge of software programming and anchors the learning about security knowledge. The second strategy is to organize underlying security knowledge in a structured manner that can stimulate learners' mental models to support more efficient learning in the specified context. The third is to guide learners to engage with concrete knowledge before studying abstract knowledge. This strategy assists learners in discovering meaningful concepts and relationships between practical functions and abstract knowledge when working in this context.

Our research attempts to place security learning in the context of real application scenarios. The benefits of this contextualized approach can also be explained by the effective mechanism of intrinsic motivation, where a learner is drawn to engage in a task because it is perceived as interesting, enjoyable, and/or useful [89, 115, 251]. Since the given context is connected and relevant to their prior knowledge and life experiences in software development, security learning can then be related to a similar programming topic that they want to learn about or a problem to be solved. We strongly believe this implies a direct effect of the contextualized learning approach on higher overall learning satisfaction, which motivates students to learn.

Our future work includes improving the usability of the user interface and enriching the knowledge content with a variety of application scenarios. We plan as well as learning experiments with bachelor students, in order to validate our proposal.

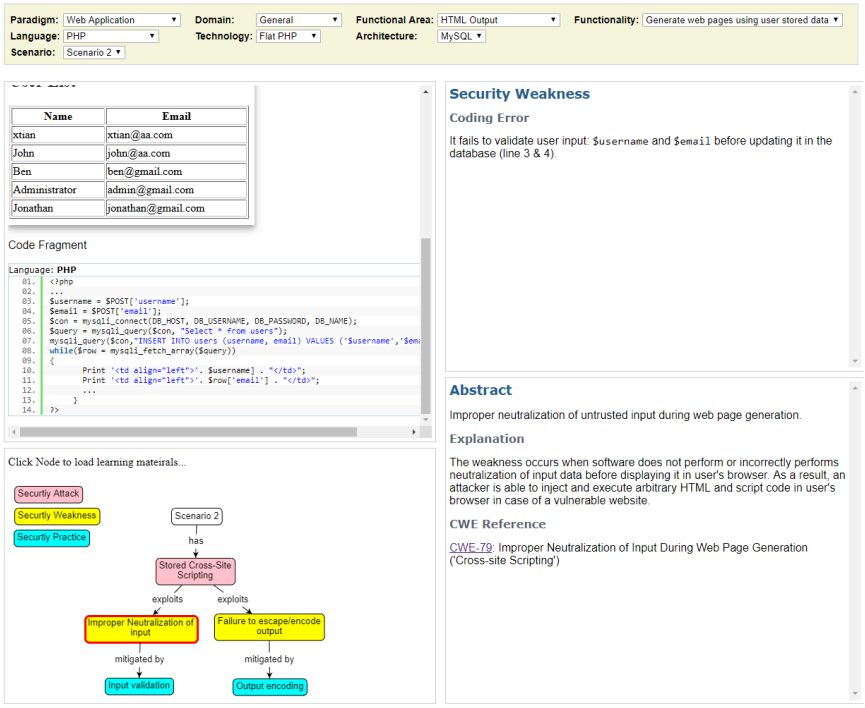


Figure 13.11: The screenshot of viewing security weakness of the scenario

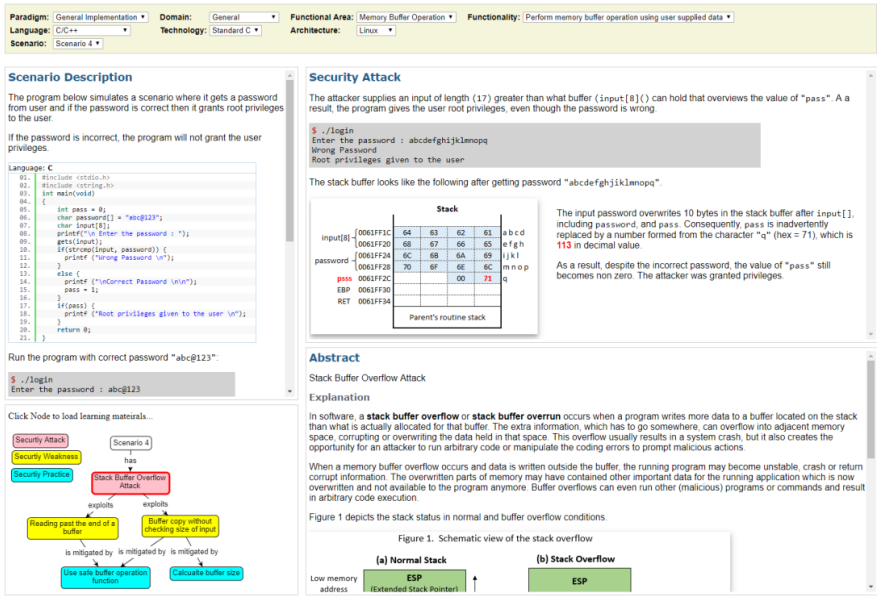


Figure 13.12: A scenario for memory buffer operations in C/C++

Chapter 14

Preliminary Evaluation of an Ontology-Based Contextualized Learning System for Software Security

Wen, Shao-Fang and Katt, Basel. "Preliminary Evaluation of an Ontology-Based Contextualized Learning System for Software Security." *In Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2019.

Author Contributions— Initial conceptualization and framework of the research were developed by Shao-Fang Wen. The research methodology and experimental design and were reviewed by Basel Katt.

Abstract—Learning software security is a big challenging task due to the vast amount of security knowledge and the difficulties in understanding the practical applications. The traditional teaching and learning materials, which are usually organized topically and security-centric, have fewer linkages with learners' experience and prior knowledge that they bring to the learning sessions. Learners often do not associate vulnerabilities or coding practices with programs similar to what they were writing in their previous time. Consequently, their motivation for learning is not touched by conventional methods. The aim of this paper is the presentation of an ontology-based learning system for software security with contextualized learning approaches, and of the results of an initial evaluation using a controlled quasi-experiment in a university learning environment. The experiment results show that the prototyped system with the proposed learning approach not only yields significant knowledge gain compared to the conventional learning approach but also gains better learning satisfaction of students.

14.1 Introduction

Software security has been a subject of a plethora of studies for at least 40 years, and a steady stream of innovations has improved software engineers' ability to secure software development and to protect applications. Improving software security requires many different approaches, such as adopting a secure software development process and security technologies. One way is to give software engineers or learners the knowledge and skills to resist attacks and handle errors appropriately [46]. To emphasize security, a relatively large number of best practices and vulnerability information have been published by security committees in publications or on the internet [313, 412, 481]. To this extent, the huge amount of information has resulted in a form of information overload for learners. Moreover, the domain of software security is quite context-specific and can be applied in diverse ways [294]. For example, the security principle of least privilege recommends that accounts should have the least amount of privilege required to perform the task. This encompasses the security practices of user rights, and resource permission such as CPU, memory, and network, which exist for specific programming languages (e.g. C, C++, PHP, Java and so on), depending on the features of the software product. As a result, learning software security becomes a complex and difficult task because learners must not only deal with a vast amount of knowledge about a variety of concepts and methods but also need to demonstrate the applicability of the knowledge through experience in order to understand their practical use.

In conventional security learning materials, the knowledge content is commonly security-centric and organized topically, which distinguishes two fundamental segments: the white-hat approach, where the main emphasis on security principles and anti-attack mechanisms, and the black-hat, which teaches how to break software and how malicious hackers write exploits. These learning materials are often described in the form of a reference manual or a guide to particular security subjects. The topical knowledge organization is useful for rote memorization of a specific security subject or for information reference later [500]; however, it is difficult for learners to understand the rationale of the topics, and correlate those topics with real software scenarios. Learners usually finish reading such materials with little understanding of the context in which the security knowledge should be applied, or with the feeling that the security domain is so extensive and software, security is so difficult to achieve that they simply cast it aside.

Our primary objective is to create conditions for more effective learning for software security that can motivate learners and stimulate their interest. This paper is part of an investigation into contextualized learning in the domain of software security, supported by empirical evaluation. We propose a learning system, which facilitates the contextual learning process by providing contextualized access to security knowledge through software application scenarios. This learning system is a place where learners can explore and relate the security knowledge to the context they are already familiar with. To develop this kind of learning system, the security

knowledge should be modeled and managed in a manner where the software security knowledge can be retrieved taking the context of the application in hand into consideration. Ontologies make it possible to give this kind of purpose since it facilitates the capture and construction of domain knowledge and enables the representation of skeletal knowledge to facilitate the integration of knowledge bases irrespective of the heterogeneity of knowledge sources [181]. In the paper, an ontology-based web application prototype is presented, which was evaluated by a preliminary experiment in the setting of a university environment. This paper also presents experimental design and results.

The rest of this paper is organized as follows: In section 14.2, we introduce the theoretical background of this study. Section 14.3 describes our design approach for software security learning. Section 14.4 presents the detailed design of an underlying ontology of the learning system. Section 14.5 describes the developed prototype while section 14.6 summarizes the experimental evaluation of the prototype. Lastly, the discussion and conclusion are presented in section 14.7.

14.2 Background

The theoretical background of this research is drawn from the field of context-based knowledge and contextualized learning. Nonaka [328] indicates that knowledge reflects a particular instance, perspective, or intention in accordance with the characteristics of a specific context, which is different from information. According to Brézillon [58, 61], knowledge comes from a variety of context and it cannot be accurately understood without context. Without proper contextual information, knowledge can be isolated from other relevant knowledge resulting in limited or distorted understanding [169]. Researchers of psychology and education indicate when knowledge is learned in a context similar to that in which the skills will actually be needed, the application of learning to the new context may be more likely [117, 122, 352]. Predmore [360] shows that learning about knowledge content within real-world experience is important because "once [students] can see the real-world relevance of what they're learning, they become interested and motivated". Since context can give guidance about when, where and why a piece of knowledge is used, considering the context in knowledge use is very necessary to enhance the applicability of knowledge [46].

Contextualized Teaching and Learning builds upon a similar concept of putting learning activities into perspective to achieve the best teaching and learning outcomes. Researchers Berns and Erickson [39] define contextualized learning as a practice that endeavors to link theoretical constructs that are taught during learning, to a practical, real-world context. Contextualized instruction in general, starts with presenting a context from which the concepts are developed on a need-to-know basis. This requires teachers to teach in a more constructivist way, i.e. to position the concepts of the learning subject in contexts recognizable to students and to stimulate the active learning of the students [346]. The contextualization of the learning on

demand can not only be seen from the point of view of an actual problem or learning situation but also in a longer-lasting process of learning activities that are integrated [425].

In computer science education, there is also a broad agreement that teaching units should start from a "real-world" context or phenomenon, aiming to create connections to prior knowledge, to increase the relevance of the material to students or to show application situations of the intended knowledge, thereby increasing motivation [120, 184]. These contrast with more traditional approaches that cover abstract ideas first, before looking at practical applications. Likewise, in software engineering, studying from a context and then abstracting the knowledge gained to be able to use it in a new context is a common way of learning programming that has been observed extensively in both new and experienced programmers [23, 243]. In order to capture and use security knowledge appropriately, it is necessary to first specify which context information is to be handled, and then represent this in a format that is understandable and acceptable to the individuals. Thus, a context for a software security topic includes the circumstances in which its technical content exists. Therefore, to talk about software security in context is to say that knowledge would not only include the basic principles and processes of software security but would consider how security knowledge is used in one or more particular domains or application areas.

14.3 Design Approach

To facilitate contextualized learning about software security and create engaging learning experiences for learners, we proposed a contextualized approach for software security learning with three strategies: (1) Starting with a meaningful scenario; (2) Stimulating learners' mental model for software security learning; and (3) Moving from concrete to abstract security knowledge. Figure 14.1 depicts an abstract representation of our design approach to the learning system for software security. The details of the strategies were described in the following sections.

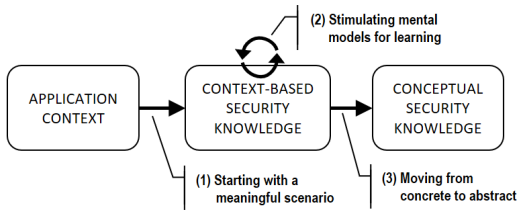


Figure 14.1: The design approach of the learning system

14.3.2 Starting with a Meaningful Scenario

Contextualized learning often takes the form of real-world examples of problems that are meaningful to the learners personally [373]. To begin the process of learning, a

meaningful situation for learners must first be established. In our study, the learning situations are created through the use of contextual scenarios in the application context, which utilize some form of anchoring situation events [85] to engage learners with security concepts that are addressed in the software problem or situation. Contextual scenarios refer to different manifestations within a context [130]. We choose a scenario-based approach because scenarios can be easily adapted to the situation of the represented applications and can be easily integrated with the conceptual security knowledge. For instance, regarding the application functionality of "Generating HTML pages" in the web application context, it includes a set of scenarios, such as generating static or dynamic pages and using external data from HTTP requests or data stores. Those scenarios can serve as anchoring events to evoke the learners' memories of programming and draw attention to software events and conditions. The use of anchor evens in our study aims to echo learners' real-world experiences to context-based security knowledge to help learners apply their emerging understandings about software security to the real software cases, thus helping them see value in their learning sessions.

14.3.3 Stimulating Mental Models for Learning

Contextual learning is a learning approach that ties brain actions in creating patterns that have meaning [113]. In order to help learners make sense of complex security knowledge and create a strong and lasting bond among security concepts while they are engaged through various anchoring events, our strategy is to elicit learners' mental models for the navigation of security knowledge. Such mental models allow learners to gain insight regarding their world by building a work scheme [160], which makes it easier for them to access the information needed to understand the knowledge domain, make predictions, and decide upon action to take [379]. Generally, our intention was to guide learners in answering What-Why-How questions while dealing with each anchoring event:

- *What are the possible attacks?*
- *Why does it encounter attacks?*
- *How can these attacks be prevented?*

Once learners answer what–why–how questions, the relationships between the security concepts are revealed in their midst, and thus, their representation of mental models expands.

14.3.4 Moving from Concrete to Abstract

To help learners gain a more flexible understanding of the study concept in a range of situations with varying levels of abstraction, we organize security knowledge by blending abstract and concrete perspectives; presenting it with a sequence from concrete to abstract. In our study, abstract knowledge refers to the conceptual security

domain knowledge while concrete knowledge relates to the contextualized scenario-specific security knowledge. The used concrete-to-abstract approach in knowledge presentation differs from the traditional, where the concepts are of foremost importance and are usually explained first before concrete examples and applications are discussed. Several studies [136, 291, 292] have shown that the majority of engineering students are sensor-type learners, who like facts, data, and observable phenomena as opposed to theoretical abstractions. As long as the concrete knowledge and the underlying abstract explanation are understood by learners, learning transfers from one context to another will be more effective.

14.4 The Underlying Ontology-Based Knowledge Model

The kernel security knowledge repository was based on ontology modeling technologies. With this model, the learning application can handle contextualized security knowledge with multiple scenarios in different application-specific contexts and integrates security concepts of security domain knowledge. Figure 14.2 shows an overview of the ontology-based knowledge model including the interrelationships of the components.

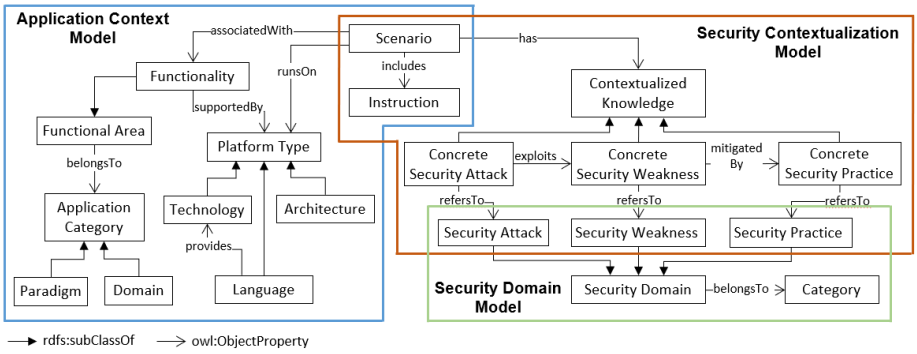


Figure 14.2: The ontology-based security knowledge model

14.4.2 Application Context Model

The context model represents a definition of what context is in a specific domain. In our ontology, the context for software security knowledge is supported by the creation of scenarios in different application contexts. The scenario presents a snapshot of possible features and corresponding code fragments in the specific functionality that is included in the Instruction class. It also draws on situated security knowledge, that is, understandings particular to the application context in which they generate. In context modeling, in addition to scenarios, we focus on characteristics that are highly relevant for retrieval within a software application.

- *Application category:* It is a set of characteristics to categorize software applications, which include two sub-classes: paradigms (e.g., web, mobile, and desktop applications, etc.) and the domains (e.g., banking, health, and logistics applications, etc.).
- *Platform type:* This superclass specifies programming languages, technologies, and architectures that are used to create the software application. Technology can be provided by a certain programming language. For example, Silverlight is the technology that has been implemented in C# language, while J2EE is the subset of Java technologies. Architectures refer to the fundamental system structure to operate the application, such as the MySQL database management system and an Android operating system.
- *Functional area:* It is a group of application functionalities, which represents an aspect of software applications that can be performed by users or other systems in a particular application category. For example, outputting HTML is a functional area in the web applications paradigm, in which generating HTML dynamically using user-supplied data is one of the functionalities. A functionality is supported and run on some combinations of platform types.

14.4.3 Security Domain Model

The security domain model describes the knowledge that is an object of teaching through a set of concepts (topics to be taught). We identify three security concepts that are most widely used throughout the security domain. Ultimately, three classes were incorporated into the security domain model: *Security Attack*, *Security Weakness*, and *Security Practice*.

- *Security Attack*: It represents actions taken against the software application with the intention of doing harm. Examples are SQL injection, Cross-Site Scripting (XSS), etc. Security attacks exploit security weakness existed in software applications.
- *Security Practice*: It represents methods, procedures or techniques to prevent security weakness. Examples are input validation and output encoding in preventing XSS.
- *Security Weakness:* It represents bug, flaws, vulnerabilities and other errors that exist in the software applications. Examples are improper to neutralize input during HTML generation and fail to perform a bound check while copying data into memory stack.

14.4.4 Security Contextualization Model

Security contextualization modeling manages security knowledge in the context of specific scenarios and brings together the conceptual knowledge that is described in the security domain model. The including security concepts are aligned with those

defined in the security domain model, which are *Security Attack*, *Security Weakness*, and *Security Practice*. However, in order to clearly state the purposes and distinguish them from the security domain model, we use different classes, namely *Concrete Security Attack*, *Concrete Security Weakness*, and *Concrete Security Practice*. The abstract class *Contextualized Knowledge* is used from which these three classes inherit common attributes such as tags or external resources. Once the domain knowledge model is defined, each security concept in the contextualized model is able to be connected to the corresponding classes in the security domain model.

14.5 The Developed Prototype

We have developed a proof-of-concept prototype to demonstrate the proposed approach. The high-level system architecture diagram is presented in Figure 14.3. The front-end was designed as a web-based user interface with PHP and JavaScript languages and through it, learners can access the knowledge content. The backend was implemented in Java and access to the ontology repository was provided through the Jena API[41], a Java framework for building semantic web applications. Jena provides extensive Java libraries for helping developers develop code that handles RDF, OWL, and SPARQL in line with published W3C recommendations[42].

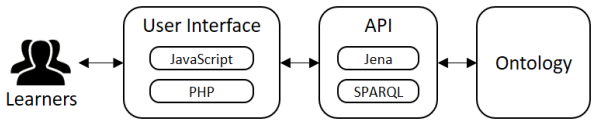


Figure 14.3: High-level system architecture diagram

In the prototype, we set up the learning environment in a web application paradigm, using a pure PHP technology and MySQL as the architectural database. To prepare for learning materials based on this specified context, the author developed a preliminary set of functionalities to operate a real web-based application, including a login module, data input/output features, data processing, and database access. Three critical application scenarios were created for each function within the scope of the application context. The user interface of the prototyped system is presented in Figure 14.4.

In the learning application, the learning process begins with the concrete in a context familiar to learners and then gradually leads to an understanding of the abstract. First of all, a meaningful situation for learners must first be established. The access to learning content in the learning application mainly happens scenario-oriented. We use the scenario as the starting point for learning security concepts on a need-to-know basis while presenting the modeled security knowledge. Based on the desired knowledge the learner selects relevant criteria from the application-context menu to

[41] https://jena.apache.org/
[42] https://www.w3.org/2001/sw/

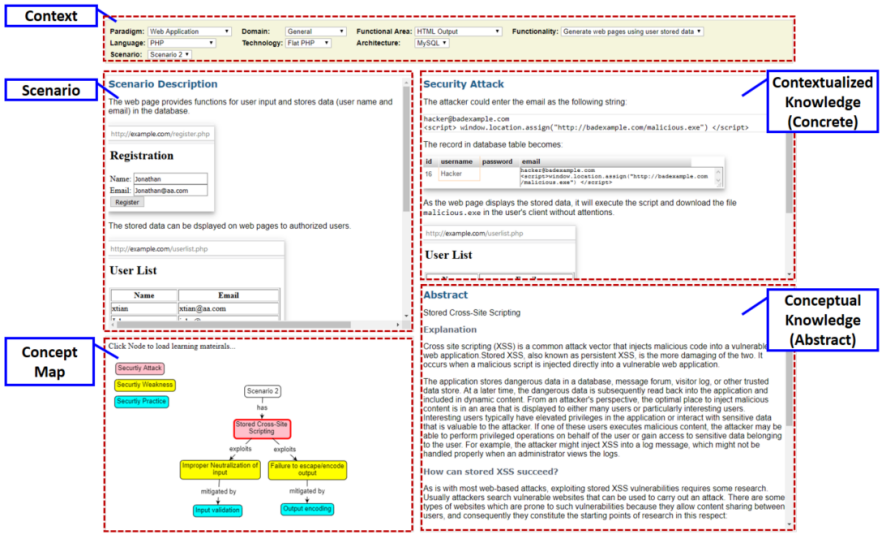


Figure 14.4: The user interface of the developed prototype

scope the learning scenario. The instructional part of the scenario is made up of practical demonstrations of the pre-described application functionality and the code fragments behind it that bridge the corresponding security knowledge.

To guide learners navigating through the contextualized knowledge efficiently, it is necessary to illustrate the relationship between the security concepts. On the one hand, it must be transparent for learners about, which causes and effects relevant to the learning content he (or she) is studying. On the other hand, this is essential for learners in order to integrate the semantical impact of the knowledge structure into the mental models for efficient learning. For the purpose, we outline the learning contents in a graphical Concept Map, which shows in the left-corner part of the screen. Concept Map is a visual representation of different concepts and their relationships. With the use of concept mapping, the learning arena can be virtualized in a learner's mind [405]. From the visual description, the reader extracts propositions and creates a mental model from the graph. Meanwhile, the extracted mental model will be inherently influenced by connecting to prior experience.

The design of our ontology is able to provide the basis for the development of the concept map of the relationship between these concepts. While a node is clicked on the concept map, the relevant knowledge content is displayed in the right half of the screen, where the upper part is the contextualized knowledge and the lower part is an abstract explanation, following the concrete-to-abstract presentation strategy. By concrete representations, we include perceptually detailed and rich materials, such as demonstrating security attacks with different exploits, identifying mistakes in the source code, and showing the secure coding practices to fix the mistakes. As described previously, the selected scenario served as an anchoring event that can be view

throughout the learning session to anchor learning in the learners' personal experience. With such a scenario presentation, learners can easily recall features of the context (e.g. code fragment) without interrupting the learning process. After experiencing the facts, learners then move on to abstract knowledge, where the conceptual explanation is presented. Therefore, dynamic, e.g., situational application scenario is integrated together with the security domain knowledge.

14.6 Prototype Evaluation

In this section, we describe the evaluation of the proposed approach as well as the developed prototype. A pre- and post-test based experiment was designed and executed in the context of the Bachelor course Software Security in Norwegian University of Science and Technology. The research design is presented in Table 14.1. The participants were 36 Bachelor students from two main study programs, IT operations in information security, and Programming.

Table 14.1: Experiment Design

Group	Number of participants	Treatment
Experiment	18	X_1
Control	18	X_2

Remark:
X_1: Learning system (Software tool)
X_2: Learning material (Hard-copy document)

The participants were randomly assigned to either control or experimental groups. The students in the experimental groups were treated the proposed learning system (X_1) while the control group adopted a conventional learning approach, which was a hard-copy document (X_2). The learning subject was focused on a common security attack in web applications: Cross-Site Scripting. According to OWASP's Top 10 Application Security Risks – 2017 [341], it is the third most risky web applications' vulnerability and the most widespread. To construct the learning material for the control group, the authors extracted information from textbooks and resources on the internet, combing with the authors' teaching experience in the domain of software security. The knowledge content was organized in the order of abstract-to-concrete where the conceptual description of the vulnerability subject was described in the first place, followed by examples with code fragments of exploits. Mitigations for the vulnerabilities were explained in the last section. Figure 14.5 shows a simplified view of the learning material X_2 for the control group.

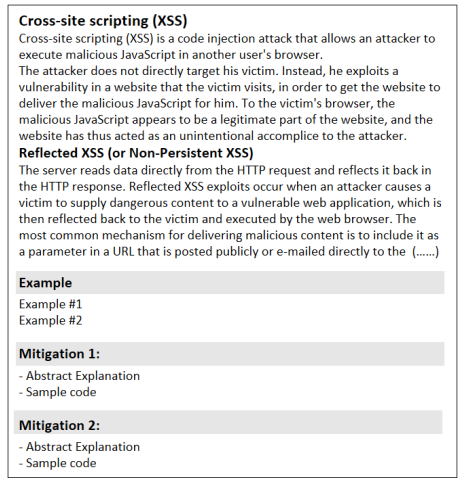

Cross-site scripting (XSS)
Cross-site scripting (XSS) is a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser.
The attacker does not directly target his victim. Instead, he exploits a vulnerability in a website that the victim visits, in order to get the website to deliver the malicious JavaScript for him. To the victim's browser, the malicious JavaScript appears to be a legitimate part of the website, and the website has thus acted as an unintentional accomplice to the attacker.
Reflected XSS (or Non-Persistent XSS)
The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a victim to supply dangerous content to a vulnerable web application, which is then reflected back to the victim and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to the (.....)
Example
Example #1
Example #2
Mitigation 1:
- Abstract Explanation
- Sample code
Mitigation 2:
- Abstract Explanation
- Sample code

Figure 14.5: A sample of the learning materials for the control group

14.7 Data Collection

To collect data and measure the dependent variables, two types of instruments were used: knowledge test sheets and the survey questionnaires. Knowledge test sheets, differentiated by pre-test (T_1) and post-test (T_2), were developed to measure the knowledge gain (i.e., T_2 to T_1), in which items were created across two types of security knowledge—theoretical and practical. Theoretical items focused on recalling and understanding of conceptual security knowledge. Practical items require students to identify possible attacks in a given software context, marking coding errors in code fragments, applying knowledge to different situations. The pre- and post-tests were similar except for the formulation of some questions, their order, and the answer options. In each test sheet, there were 12 questions and the value for each question was five points.

We designed two survey questionnaires (S_1 and S_2) to collect students' perceptions of the two learning approaches. Questionnaire S_1 was developed to measure the learning satisfaction of students in the experimental group. Two major sections with five questions for each were designed in S_1, which are "System operation" and "Learning attitude". In this questionnaire, all respondents were required to choose the answer that reflects their own views and stance on the statements that are administered in accordance with the 5-point Likert scale, ranging from "strongly disagree" to "strongly agree". Questionnaire S_2 was created to collect all students' perceptions of the two approaches (i.e., X_1 vs. X_2) in order to understand their learning preferences. In this questionnaire, students were asked to indicate their preferred learning approach that best fits the statement of each question.

14.8 Experimental Procedure

The detailed experimental procedure is presented in Table 14.2. The students were randomly assigned into two groups (experimental and control group) while they entered the classroom. They were first introduced to the main objectives of the experiment and informed of the procedure. After completing the pre-test sheets, students went through and studied the learning materials using the treatments assigned to them. At the end of the learning session, all students took the post-test exam where students of the experiment group filled out questionnaire S_1 additionally. In the last 30 minutes, students were asked to experience the learning approaches that were different from the previous one they practiced, and completed questionnaire S_2 afterward. This ended the experimental procedure.

Table 14.2: The experimental procedure

Step	Activity	Duration (minutes)	Treatment	
			Experimental group	Control Group
1	Pre-test	15	T_1	T_1
2	Learning session	60	X_1	X_2
3	Post-test	15	T_2	T_2
4	Survey I	5	S_1	--
5	Experiencing	25	X_2	X_1
6	Survey II	5	S_2	S_2

Remark:
T_1: Test sheet (Pre-test)
T_2: Test sheet (Post-test)
X_1: Learning system
X_2: Learning material
S_1: Questionnaire I
S_2: Questionnaire II

14.9 Experimental Analysis

14.9.1 Knowledge Gain Analysis

The students' knowledge gain on the different type of treatment were determined using the compare means analysis. Table 14.3 reveals the mean analysis of students' performance on the pre- and post-test, including the mean scores and standard deviation. The results of the statistical analysis show that there was a positive knowledge gain (i.e. post-test to pre-test score) for both groups. However, the experimental group had higher achievement levels than the control group, as shown in Figure 14.6. The average knowledge gain in the control group was 10.28 whereas it was 16.11 in the experiment group.

Table 14.3: Compared means analysis of students' performance on the pre- and post-test

		N	Mean	Std. Deviation
Control Group	Pre-Test	18	30.00	11.757
	Post-Test	18	40.28	9.922
Experimental Group	Pre-Test	18	30.83	11.789
	Post-Test	18	46.94	7.503

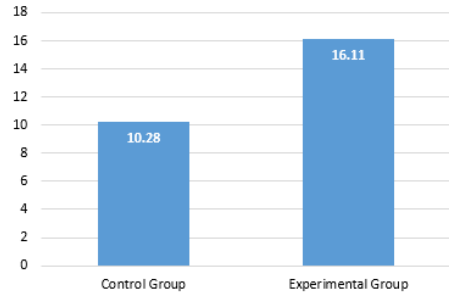


Figure 14.6: Knowledge gain for the control and experiment groups

To determine whether there was a significant difference between the pre-test performances of the experimental and control groups, an independent sample t-test was used. Table 14.4 shows the t-test analysis for pre-test results. The significant level (0.519) of Levine's test for equal variance was greater than 0.05, indicating "Equal variance assumed". Following the value indicated in Levine's test, we got "Sig. (2-tailed)" value of 0.833, which is above 0.05. Therefore, the null hypothesis of the independent sample t-test was rejected ($p > 0.05$). This implies that there was no significant difference between the two groups in terms of pre-test scores (i.e., the initial security knowledge).

Table 14.4: Independent sample t-test for pre-test score

		Levene's Test		t-test for Equality of Means				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Pre-Test	Equal variances assumed	0.425	0.519	-0.212	34	0.833	-0.833	3.924
	Equal variances not assumed			-0.212	34	0.833	-0.833	3.924

We also performed an independent sample t-test for the post-test mean scores. As can be seen in Table 14.5, the difference between the post-test mean score of the two groups was significant (2-tailed Sig. = 0.029, $p < 0.05$). This indicated that the experimental treatments have resulted in a significant difference in security knowledge gain between the two groups of students.

Table 14.5: Independent sample t-test for the post-test score

		Levene's Test		t-test for Equality of Means				
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference
Post-Test	Equal variances assumed	0.142	0.709	-2.274	34	0.029	-6.667	2.932
	Equal variances not assumed			-2.274	31.651	0.030	-6.667	2.932

Then in order to see whether the treatment given to the experimental group had caused a statistical difference in students' performances; a paired sample t-test was performed as well. Table 14.6 shows that there was a significant average difference between pre-test and post-test scores ($t_{17} = 7.734$, $p < 0.05$) in the experiment group. Therefore, the significance of the knowledge gain in the experimental group can be concluded.

Table 14.6: Paired sample t-test of pre- and post-test for the experimental group

		Paired Differences					
		Mean	Std. Deviation	Std. Error Mean	t	df	Sig. (2-tailed)
Experiment Group	Post-test - Pre-test	16.111	8.838	2.083	7.734	17	0.000

14.9.2 Questionnaire Analysis

Table 14.7 presents the evaluation of students' learning satisfaction (questionnaire S_1) in the experimental group. As shown in the table, the satisfaction degree achieved 4.07 in terms of system operation and 4.09 regarding the learning attitude.

Table 14.8 summarizes the result of students' learning preferences evaluation (questionnaire S_2) for the two learning approaches. It indicates that among the 36 students, 77.78% of students agreed that the learning system organized security knowledge that fit their learning preferences. Meanwhile, 88.89% of students considered the contextualized learning system can promote their learning interest much more than the conventional materials. The most important, all students thought that the proposed learning system could ease information overload on learning security subjects.

Table 14.7: The evaluation of student' learning satisfaction in the experimental group

Category	Question	Mean
System Operation	• I agree that the applied learning technique in the system is novel and it can assist my learning.	4.11
	• I am very clear about the learning procedure embedded in the system.	4.00
	• The system organizes security knowledge in a structured and collected manner.	4.21
	• The knowledge content provided by the system is easy to understand.	4.00
	• I think that the system is useful for learning security knowledge.	4.05
	Average	4.07
Learning Attitude	• The system helps me deepen the memorized impression of the learning subject.	4.11
	• The system helps me relate security knowledge to what I knew or experienced before.	4.16
	• The system reduces the difficulty of learning secure programming.	4.11
	• I find that at times studying the learning materials gives me a feeling of personal satisfaction.	4.05
	• The system helps me foster a positive attitude toward learning security knowledge.	4.00
	Average	4.09

Table 14.8: The evaluation of student' learning preferences

Question	Proposed Learning System (%)	Conventional Material (%)
• The approach organizes security knowledge in a way that fits my learning preference.	77.78	22.22
• The approach can promote my learning interest much more.	88.89	11.11
• The approach eases information overload on learning security subjects.	100	0
• The approach can make my security knowledge progress more.	72.22	27.78
• The approach can benefit most people in learning software security.	83.33	16.67

14.10 Discussion and conclusion

In this study, an ontology-based contextualized design approach of the software-security learning system is proposed with three strategies. The first is to establish meaning scenarios to create a meaningful situation for learners. The design of the application context aims to activate the learner's prior knowledge of software programming and anchors the learning about security knowledge. The second strategy is to organize underlying security knowledge in a structured manner that can stimulate learners' mental models to support more efficient learning in the

specified context. The third is to guide learners to engage with concrete knowledge before studying abstract knowledge. This strategy assists learners in discovering meaningful concepts and relationships between practical functions and abstract knowledge when working in this context.

The developed prototype was evaluated by a controlled experiment with 36 bachelor students. We used pre-test/post-test to measure students' security knowledge gain, and questionnaires to evaluate their learning satisfaction. The result of the pre-test/post-test experiment indicates an increase in students' level of security knowledge for both learning approaches; the experimental group yielded more knowledge gain on average than the control group. According to the statistical t-test analysis result, there is no significant difference between the two participating groups of students in terms of initial security knowledge (Table 14.4). However, there resulted in a statistical difference in security knowledge gain between the two groups of students after applying the treatments (Table 14.5). Additionally, the average difference between pre-test and post-test scores for the experiment group is also proved significant (Table 14.6). This concludes that students using the proposed learning system yielded significantly better knowledge gain than those using conventional learning materials.

On the other hand, the evaluation of students' satisfaction with the two learning approaches shows a positive result, as the respondents expressed their higher learning satisfaction with the learning system using contextualized security knowledge than conventional learning materials. The survey results also show that most students were very interested in the proposed learning system and all agreed that this approach could ease the information load effectively. Our approach attempts to place security learning in the context of real application scenarios. The benefits of this contextualized approach can also be explained by the effective mechanism of intrinsic motivation, where a learner is drawn to engage in a task because it is perceived as interesting, enjoyable, and/or useful [89, 115, 251]. Since the given context is connected and relevant to their prior knowledge and life experiences in software development, security learning can then be related to a similar programming topic that they want to learn about or a problem to be solved. We believe this implies a direct effect of the contextualized learning approach on higher overall learning satisfaction, which motivates students to learn.

Although the present approach seems to be effective, there are some limitations in generalizing the findings of this study. First, the findings were from an experiment in a real classroom setting of a Software Security course at a university; therefore, it could be difficult to generalize the finding to other learning environments or courses. Second, it was based on a relatively small group of subjects (36 students). There is a need to expand the number of participants. Third, since this study evaluated the outcomes immediately after a short-term learning session (1 hour), it is not certain what the knowledge retention is and for how long it will be retained. It is suggested to evaluate the effectiveness of the learning system over a long-term period.

In conclusion, our proposed approach to establishing a contextualized learning system does provide a sounder basis for software security learning than conventional methods. Consequently, our study produced promising results, which may be of value for educational practice. It is recommended that curriculum developers of software security materials should use the context-based approach as one of the teaching strategies to improve students' performance in security knowledge. As part of our future work, we plan to improve the usability of the user interface and to enrich the knowledge content with a variety of application scenarios; meanwhile, extensive experiments can be conducted to further evaluate the effectiveness and benefits of this approach, including long-term evaluation. In addition, it would be interesting to investigate the learning performance of learners with different learning environments, such as software-project team training and self-directed learnin

Chapter 15

Learning Software Security in Context: An Evaluation in Open Source Software Development Environment

Wen, Shao-Fang and Katt, Basel. "Learning Software Security in Context: An Evaluation in Open Source Software Development Environment." *In Proceedings of the 14th International Conference on Availability, Reliability, and Security*. ACM, 2019.

Author Contributions— Initial conceptualization and framework of the research were developed by Shao-Fang Wen. The research methodology and evaluation design and were reviewed by Basel Katt.

Abstract— Learning software security has become a complex and difficult task today than it was even a decade ago. With the increased complexity of computer systems, it is hard for software developers to master the expertise required to deal with the variety of security concepts, methods, and technologies that are required in software projects. Although a large number of security learning materials are widely available in books or open literature, they are difficult for learners to understand the rationale of security topics and correlate the concepts with real software scenarios. To tackle this learning issue, our research is focused on forging a contextualized learning environment where learners can relate the learned security knowledge to the context that they are familiar with. In this paper, we present our evaluation study of the learning system in the open source software development environment. The results demonstrate that contextualized learning can help OSS developers identify their necessary security knowledge, improve learning efficiency and make security knowledge more meaningful for their software development tasks.

15.1 Introduction

Security has become an important part of today's software development projects. Improving software security requires that software engineers acquire relevant knowledge and skills to secure software development such that they can resist attacks and handle security errors appropriately [46]. However, learning software security has become a complex and difficult task today than it was even a decade ago [459]. Nowadays, with the increased complexity of computer systems and a variety of applications, the intricacy of software development projects have been grown consistently. Each software product and process is different in terms of goals and contexts. It is hard for software developers to master the expertise required to cope with the variety of security concepts, methods, and technologies that are required in software projects. Developers are often exposed to this diversity, which makes the software discipline inherently experimental [33, 275].

On the other hand, security knowledge can be both dynamic and situation-specific [294], and the complexity of knowledge usually exceeds the capacity of individuals to solve problems by themselves. Learners must not only cope with a variety of security attacks and countermeasures but also have to demonstrate the applicability of the knowledge countermeasures through experience in order to understand their practical use. Although much security information is widely available in the form of checklists, standards, and best practices in books, open literature or on the Internet [313, 412, 481], it remains difficult for software engineers to correlate relevant pieces of security knowledge to apply to their application-specific situations. There remains a lot of confusion in learners' minds as to the rationale of security topics. We argue that the traditional approach, which usually organizes knowledge content topically, with security-centric, is not suitable to motivate learners and stimulate their interest. Developers or security learners often feel that the security knowledge is such extensive and software security is so difficult to achieve, that they simply cast it aside.

Keeping in view of the aforementioned facts, our position is that security knowledge should be contextualized and placed in a meaningful situation that makes sense to the learners to enhance their understanding and make the concepts more relatable. As Gary McGraw points out, the domain of software security is rather context-specific, and the real project situation is necessary to apply the security concepts within the specific system [294]. Researchers have also indicated that studying from a context and then abstracting the knowledge gained to be able to use it in a new context is a common way of learning programming that has been observed extensively in both new and experienced programmers [23, 243]. In computer science education, there is also a broad agreement that teaching units should start from a "real-world" context or phenomenon, aiming to create connections to prior knowledge, to increase the relevance of the material to students or to show application situations of the intended knowledge, thereby increasing motivation [120, 184].

To this end, our research is focused on forging a software security learning environment where learners can explore security knowledge and relate it to the context that they are familiar with. We have proposed a learning system for software security with a context-based learning approach, which adaptively places security knowledge in the appropriate context of software development. We have previously carried out two evaluations for the proposed learning approach and the learning tool in a university learning environment [488, 489]. The experiments showed that both the context-based learning approach and the developed tool not only yielded significant knowledge gain compared to the conventional approach but also gains better learning satisfaction of students. As part of an investigation into contextualized learning in the domain of software security, we are also interested to discover and examine the impact of the learning approach in real software-project environments. In this paper, we present our evaluation study in the open source software (OSS) development environment. Our results demonstrate that contextualized learning can help OSS developers identify their necessary security information, improve learning efficiency and make security knowledge more meaningful for their software development tasks.

The paper is organized as follows. After the introduction, we introduce the theoretical background of this study in section 15.2. Section 15.3 describes the proposed contextualized learning system. In section 4, we describe the method of the evaluation study. Section 15.5 presents the result of the evaluation. In section 15.6, we discuss the results. Lastly, the conclusion is presented in section 15.7.

15.2 Contextualized Learning

Contextualized Teaching and Learning builds upon a similar concept of putting learning activities into perspective to achieve the best teaching and learning outcomes. Researchers Berns and Erickson define contextualized learning as a practice that endeavors to link theoretical constructs that are taught during learning, to practical, real-world context [39]. The underlying theme behind contextual learning activities is simple. It recognizes that by embedding instructions in contexts that adult learners are familiar with, learners more readily understand and assimilate those instructions. Naidu [319] also points out that learning is most effective when learners work on realistic problems with guidance. The contextualized experience helped them develop a deeper understanding that positioned them to better comprehend the abstract idea, and see how it manifested in actual contexts [161].

Contextualized instructions, in general, starts with presenting a context from which the concepts are developed on a need-to-know basis [38]. This requires teachers to teach in a more constructivist way, i.e. to position the concepts of the learning subject in contexts recognizable to students and to stimulate the active learning of the students [346]. The contextualization of the learning on demand can not only be seen from the point of view of an actual problem or learning situation but also in a longer-lasting process of learning activities that are integrated [425]. Therefore, a context for

a software security topic includes the circumstances in which its technical content exists. To talk about software security in context is to say that knowledge would not only include the basic principles and processes of software security but would consider how security knowledge is used in one or more particular domains or application areas.

The concept of learning in context has been widely addressed in education and psychology literature over the years, and the effectiveness of contextualized learning has been demonstrated in the setting of interactive school classrooms. However, it is still unclear how this concept can be synthesized and applied in the domain of software security. Our study aims to mitigate this research gap by delivering a tool-based contextualized learning approach to facilitate software security learning in a way that can motivate learners.

15.3 Contextualized Learning System for Software Security

15.3.1 Concepts

The basic concept of the contextualized learning system is to facilitate the contextual learning process by providing contextualized access to security knowledge through real software application scenarios. To develop this kind of learning system, we first proposed a context-based learning approach to regulate the contextualized learning process about software security. Following the proposal of the learning approach, we designed the kernel ontology-based knowledge repository and the system user interfaces. Figure 1 depicts the design consideration of the contextualized learning system. We introduce our proposed learning approach for software security and the underlying ontological security knowledge model in the below sections.

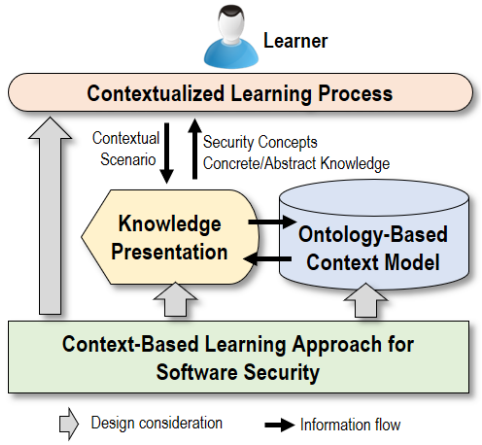


Figure 15.1: The design concept of the proposed security learning system

15.3.2 Context-Based Learning Approach

To facilitate contextualized learning about software security and create engaging learning experiences for learners, we proposed a contextualized approach for software security learning with three strategies.

A. Starting with a Meaningful Scenario

Contextualized learning often takes the form of real-world examples of problems that are meaningful to the learners personally [373]. Creating the relevance of the learning knowledge before going into the details could provide a stronger foundation for the learning process. Therefore, to begin the process of learning, a meaningful situation for learners must first be established. In our study, the learning situations are created through the use of contextual software scenarios, which refer to different manifestations within an application context. We choose a scenario-based approach because scenarios can be easily adapted to the situation of the represented applications and can be easily integrated with the contextualized security knowledge. In essence, this scenario-based strategy draws on situated knowledge - that is, understandings particular to the software problems or situations in which they are generated. At the same time, scenarios, inherently possess the dramatic potential to optimize learning processes and outcomes.

B. Stimulating Mental Models for Learning

Contextual learning is a learning approach that ties brain actions in creating patterns that have meaning [113]. In order to help learners make sense of complex security knowledge and create a strong and lasting bond among security concepts while they are engaged through various anchoring events, our strategy is to elicit learners' mental models for the navigation of security knowledge. Such mental models allow learners to gain insight regarding their world by building a work scheme, which makes it easier for them to access the information needed to understand the knowledge domain, make predictions, and decide upon action to take [379]. In order to be useful explanatorily, a mental model has to have a similar relation-structure to the reality it models. Then the constructed mental model can be used to answer questions or solve problems [235]. Generally, our intention was to guide learners in answering three questions while dealing with each software scenario:

a. *What are the possible attacks?*
b. *Why does it encounter attacks?*
c. *How can these attacks be prevented?*

The knowledge structure serves as the basis for both knowledge retention and retrieval, as well as transfer. Once learners answer what–why–how questions, the relationships between the security concepts are revealed in their midst, and thus, their representation of mental models expands.

C. Moving from Concrete to Abstract

To help learners gain a more flexible understanding of the study concept in a range of situations with varying levels of abstraction, we organize security knowledge by blending abstract and concrete perspectives; presenting it with a sequence from concrete to abstract. The used concrete-to-abstract approach in knowledge presentation differs from the traditional, where the concepts are of foremost importance and are usually explained first before concrete examples and applications are discussed. In such a concrete-to-abstract knowledge presentation, learners discover meaningful relationships between practical functions and abstract knowledge in the context of real applications. Psychologists and educators have indicated that abstract understanding is most effectively achieved through experience with perceptually rich, concrete representations [171], while concrete materials make concepts real and therefore easily internalized [226]. As long as the concrete knowledge and the underlying abstract explanation are understood by learners, learning transfers from one context to another will be more effective.

15.3.3 The Underlying Ontology

The role of the ontology in this learning system is to provide a vocabulary for representing knowledge about the software security domain and for providing linkages with specific situations in the application context. Ontologies facilitate the capture and construction of domain knowledge and enable the representation of skeletal knowledge to facilitate the integration of knowledge bases irrespective of the heterogeneity of knowledge sources [181]. Figure 15.2 shows the ontology-based knowledge model, which consists of three sub-models: the application context model, the security domain model, and the security contextualization model. With this model, the learning system can handle contextualized security knowledge with multiple scenarios in different application-specific contexts and integrates security concepts of security domain knowledge.

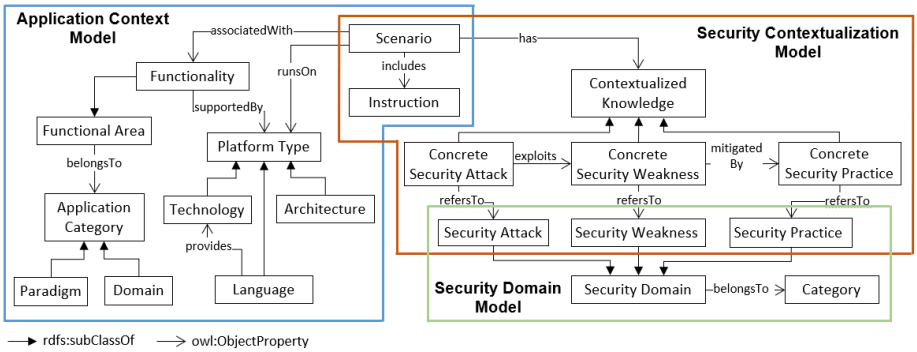


Figure 15.2: An overview of the ontology-based security knowledge model

A. Application Context Model

The context model represents a definition of what context is in a specific domain. In our ontology, the context for software security knowledge is supported by the creation of scenarios in different application contexts. The scenario presents a snapshot of possible features and corresponding code fragments in the specific functionality that is included in the Instruction class. It also draws on situated security knowledge, that is, understandings particular to the application context in which they generate. In addition to scenarios, we focus on characteristics that are highly relevant for retrieval within a software application, concerning three perspectives:

a. The application category that scenario/functionality belongs to,
b. The platforms that the scenario functionality used, and
c. The functional area (and the corresponding functionalities) that the application associated with.

B. Security Domain Model

The security domain model describes the knowledge that is an object of teaching through a set of concepts (topics to be taught). To design a security knowledge structure (schema) that is easier to store in the learners' memory for learning, the schema should be simplified and kept to the point for reducing the content load. Therefore, we identify three security concepts that are most widely used throughout the security domain. Ultimately, three classes were incorporated into the security domain model: *Security Attack*, *Security Weakness*, and *Security Practice*. From a security domain point of view, we only want to indicate which principles or abstract ideas are needed, not their practical implementation. Therefore, we describe security knowledge in this model at a level of abstraction. The instances of these classes specify only the fundamental characteristics of the security concepts, not specific software application aspects. The main advantage of this design is to share a common understanding of the conceptual security knowledge among different security contexts.

C. Security Contextualization Model

The term contextualization is used here to describe the process of drawing specific connections between security domain knowledge being taught and an application context in which the domain knowledge can be relevantly applied or illustrated. To this extent, the security contextualization modeling manages security knowledge in the context of specific scenarios and brings together the conceptual knowledge that is described in the security domain model. The including security concepts are aligned with those defined in the security domain model, which are *Security Attack*, *Security Weakness,* and *Security Practice*. However, in order to clearly state the purposes and distinguish them from the security domain model, we use different classes, namely *Concrete Security Attack*, *Concrete Security Weakness,* and *Concrete Security Practice*. The

abstract class *Contextualized Knowledge* is used from which these three classes inherit common attributes such as tags or external resources. Once the domain knowledge model is defined, each security concept in the contextualized model is able to be connected to the corresponding classes in the security domain model.

15.4 Implementation

15.4.1 System Architecture

The general architecture of the system is presented in Figure 15.3. The front-end of the system was designed as a web-based user interface with HTML and JavaScript libraries: JQuery[43] and GoJS[44]. The backend was implemented in Virtuoso[45] and using Jena[46] API for accessing to the ontology repository. Virtuoso is a cross-platform hybrid data server that combines SQL, XML, Resource Description Framework (RDF), and free-text data management with the functionality of a web application server in a single system.

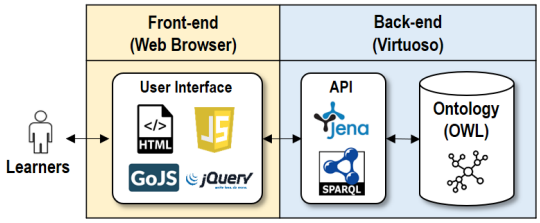


Figure 15.3: System architecture diagram

To construct the ontology, we used Protégé Editor and Web Ontology Language (OWL)[47]because of its simplicity and popularity [444]. When searching the ontology, we use SPARQL[48] protocol to extract information from the RDF. We installed a Virtuoso server and uploaded the ontology (i.e., OWL files) into the server as Linked Data using the quad store upload feature of Virtuoso. That is, the OWL files are stored in the form of Linked Data to deploy on the Web via the ontology query language (i.e., SPARQL). Jena is a Java framework for building semantic web applications, which provides extensive Java libraries for helping developers develop code that handles RDF, OWL, and SPARQL in line with published W3C recommendations[49].

[43] https://jquery.com/
[44] https://gojs.net/latest/index.html
[45] https://virtuoso.openlinksw.com/
[46] https://jena.apache.org/
[47] https://www.w3.org/OWL/
[48] https://jena.apache.org/tutorials/sparql.html
[49] https://www.w3.org/2001/sw/

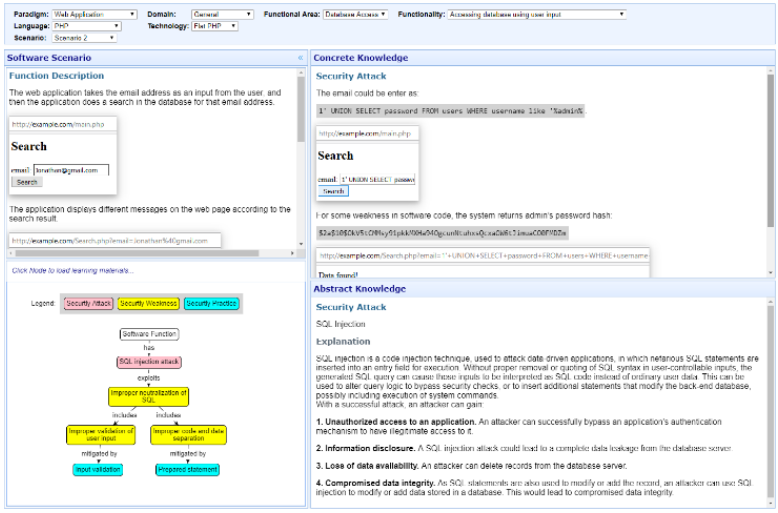
15.4.2 System Features

The system features are shown in Figure 15.4(a) and (b), in which two different scenarios are demonstrated: "Accessing database using user input" (a) and "Performing operations on a memory buffer." (b) The prior scenario belongs to the paradigm of Web Application with PHP programming language while the latter is under General Implementation with C/C++ language.

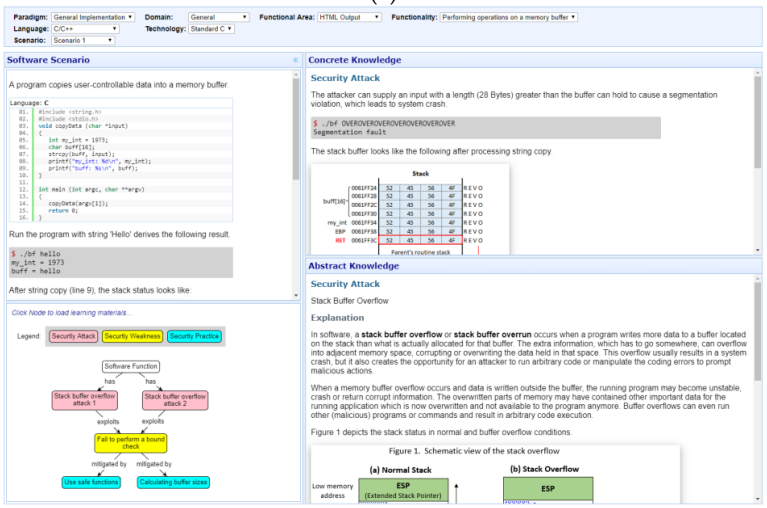
Figure 15.5 illustrates how learners are guided by the learning process of the system. The learning process begins with a selected contextualized scenario in the application context familiar to learners and then gradually leads to an understanding of the abstract part of security knowledge. First, the learner defines criteria from the application-context menu to scope the learning session based on his (or her) desired knowledge. The instructional part of the scenario is made up of practical demonstrations of the pre-described application functionality and the code fragments behind it.

To guide learners navigating through the contextualized knowledge efficiently, we outline the knowledge contents in a graphical Concept Map, developed using GoJS. Concept Map is a visual representation of different concepts and their relationships. The contextualized concept map demonstrates how security knowledge can be made more relevant to the linkage of real-world items by demonstrating their relationships. With the use of concept mapping, the learning arena becomes transparent and can be virtualized in a learner's mind [405]. This transformation is essential for learners in order to integrate the semantical impact of the knowledge structure into the mental models for efficient learning.

While a node is clicked on the concept map, the knowledge content correspondent to this concept is displayed in the right half of the screen, where the upper part is the contextualized knowledge and the lower part is the abstract explanation, following the concrete-to-abstract presentation strategy. By concrete representations, we include perceptually detailed and rich materials, such as demonstrating security attacks with different exploits, identifying mistakes in the source code, and showing the secure coding practices to fix the mistakes. With the scenario instruction displaying aside, learners can easily recall the demonstrations of the software functions without interrupting the learning process. After experiencing the facts, learners then move on to the section of abstract knowledge, where the corresponding conceptual knowledge is presented. In such an environment, learners discover meaningful relationships between the abstract explanation and the practical demonstration in the context of real software applications; security concepts are internalized through the process of discovering, reinforcing, and relating.



(a)



(b)

Figure 15.4: Snapshots of the contextualized learning system

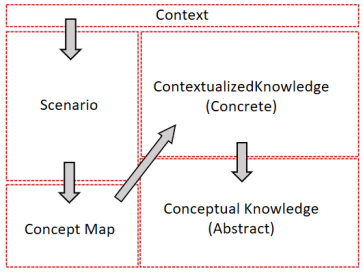


Figure 15.5: The embedded learning process in the system

15.5 Study Method

To evaluate the efficacy of the proposed security learning system, a questionnaire-based survey was conducted to collect OSS developers' perception of the proposed learning approach and system features.

15.5.1 Study Setup

In preparation for the study, we identified two common software vulnerabilities in web applications: SQL Injection (SQLi) and Cross-Site Scripting (XSS) as the learning subjects. SQLi and XSS were among the OWASP's Top 10 [341] most critical web application vulnerabilities in the past decade. For preparing the ontology of the system, we first set up the learning environment in a web application paradigm, an e-Store. For this specified context, the author developed two sets of functionalities to operate a web-based e-Store application using two different programming languages: PHP and Java, including a login module, data input/output features, data processing, database access, and payment functions. Three scenarios were manipulated under critical functionalities to demonstrate the two vulnerabilities within the scope of the e-Store system, including the corresponding vulnerable code fragments, exploits, and mitigations. With the readiness of the real software scenarios, we then constructed all learning materials and filled the ontology via Protégé application.

15.5.2 Data Collection

This study was designed to examine the potential of adopting the idea of a context-based learning system for software security for OSS developers. For the purpose, the use of a survey is deemed appropriate in this study, as the survey enables clear, direct, and objective answers to the questions presented to the respondents [40]. In this study, a self-administered web-based questionnaire was used to collect individual-level perception data from participants in OSS projects. The purpose of the questionnaire was to validate the learning system by eliciting respondents' perceptions and opinions of the learning approach and system features that support software-security learning in OSS projects. The survey instruments, which consisted of four sections, were created and hosted using Google Forms. Section 1 addressed demographics information of participants. In section 2, respondents were asked to rate the system features (Table 15.1), ranging from "very impractical" to "very practical", administered in accordance with the 5-point Likert scale. Section 3 dealt with the learning approaches embedded in the system. Respondents were required to choose the answer that reflects their own views and stance on the statements which were ranged from "strongly disagree" to "Strongly agree", with a 5-point Likert scale (Table 15.2). In the last section, participants were allowed to share their thoughts or suggestions on all aspects of the learning system.

Table 15.1: Evaluation items for system features

Evaluation Item	Question
Software Scenario	• The system introduces security subjects using common software functions.
Concept Map	• The system uses a graphical concept map to outline the knowledge content.
Security Concepts	• The system forms the main theme of security learning using three concepts: Security Attack, Security Weakness, and Security Practice.
Contextualized Knowledge	• The system demonstrates practical security knowledge in connection with the scenario.
Concrete-to-Abstract	• The system guides learners studying concrete/practical security knowledge first, then the abstraction/theory.

Table 15.2: Evaluation items for the learning approach

Evaluation Item	Question
Effectiveness	• This system can effectively assist learners in obtaining software security knowledge.
Difficulty reduction	• The learning approach reduces the difficulty of learning software security.
Experience correlation	• The approach helps me relate security knowledge to what I knew or experienced before.
Interest Promotion	• The approach promotes my interest in learning software security.
Learning Preference	• The system guides learners studying concrete/practical security knowledge first, then the abstraction/theory.

15.5.3 Participants

For the setup of this study, we recruited OSS developers on GitHub by sending out a research invitation between January 2019 and February 2019. All data collected through the survey was non-identifiable. The email invitation included an introduction to the research and links to the learning system and to the survey site. The only participation requirement of participants was the experience of web application development. A total of 21 voluntary participants accepted the invitation and completed the questionnaire after trying out the system. GitHub is an online database of OSS projects. As of June 2018, GitHub reported more than 30 million users [164] and 57 million repositories [163], making it the largest host of source code in the world.

15.6 Result

15.6.1 Respondent Demographics

Table 15.3 describes the general demographic information of the 21 respondents, in terms of gender, age and seniority in OSS development. 90% of respondents were male, while there were only 2 female respondents. A large body of participants, that is 85%, was between 20 and 40 years old and over 70% of respondents had over 3 years of experience in OSS development. As shown in Figure 15.6, Java, Python, and PHP are the top 3 programming languages that most respondents are familiar with in this study.

Table 15.3: Demographic analysis of the respondents (n= 21)

Item	Category	Frequency	Percentage
Gender	Male	19	90.48%
	Female	2	9.52%
Age	<20	1	4.8%
	20–30	12	57.1%
	31–40	6	28.6%
	41–50	2	9.5%
Seniority in OSS development	6 months to 1 year	1	4.8%
	1 to 3 years	5	23.8%
	3 to 5 years	9	42.9%
	More than 5 years	6	28.6%

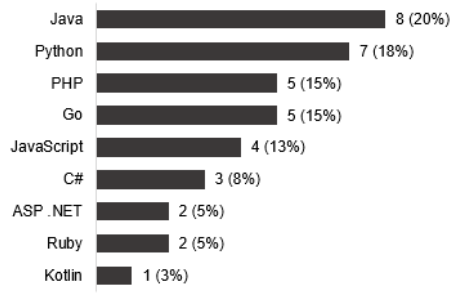


Figure 15.6: The distribution of programming languages that the respondents are familiar with

15.6.2 Satisfaction Analysis for System Features

The mean scores of the system features are plotted as a radar chart with five axes (Figure 15.7) according to each evaluation item. As can be seen from the chart, the mean scores of the system features ranged from 4.00 (for Contextualized knowledge) to 4.67 (for Concept map). The highest rating category made by the respondents was "Concept map". Most of the respondents expressed that the design of the Concept

map was attractive and thought it was useful to guide the learning process. They commented:

"I like the color-design concept. Neat and simple. Easy to follow."

"Have a node graph that helps me a lot to see stuff, not in paragraph form, but to capture the cause and effect."

"The sense of connecting security problems and solutions is really good."

Respondents also recognized the use of real software scenarios in introducing security knowledge. One respondent stated:

"When I learn [software] security, I have a very fuzzy view, to begin with, and then I kind of work at it read about it, and wait for the lightbulb to go on. I think [to start with] cases help me turn those lightbulbs on immediately."

In addition, most also appreciated the arrangement of contextualized and abstract security knowledge in the system. Some of the comments were indicated below:

"[…] clear and concise. Straight to the point, easy to understand"

"That way the sample code and the description are put together helps me learn the [security] concepts."

15.6.3 Satisfaction Analysis for the Learning Approach

We carried out reliability tests using IBM SPSS software by calculating Cronbach's alpha to examine the internal consistency of the five evaluation items within the category of "Learning approach", and determine the scale in questions is unidimensional (Figure 15.8). The derived alpha value was 0.834, which was above the acceptable threshold (0.70) suggested by Nunnally [332]. Thus, the survey items on the instrument are deemed highly reliable and appropriate for such research.

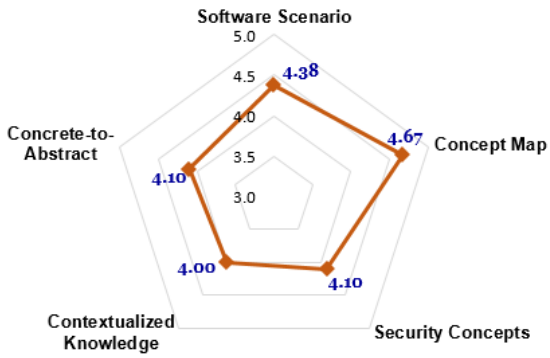


Figure 15.7: Radar chart showing the mean score of system features

Case Processing Summary

		N	%
Cases	Valid	21	100.0
	Excluded[a]	0	.0
	Total	21	100.0

a. Listwise deletion based on all variables in the procedure.

Reliability Statistics

Cronbach's Alpha	N of Items
.834	5

Figure 15.8: SPSS reliability test of evaluation items within the category of "Learning approach"

To understand respondents' perceptions regarding the learning approaches embedded in the system, we carried out a descriptive statistical analysis for the five survey items. Table 15.4 shows the analysis result, including the frequency of the valid values, means, and standard deviation. The result shows that mean scores for the five survey items all reached 4, indicating a high overall satisfaction for the learning approach expressed by the respondents. To obtain a closer view of the respondents' perception with our proposal, we depicted the proportion of responses of each survey item in Figure 15.9. From the perspective of simplicity learning, the vast majority of respondents (91%) expressed their agreement that the learning approach can reduce the difficulty of learning software security. In line with this, 85% of respondents agreed that the leaning approach creates conditions for effective leering about software security. In addition, over 80% of respondents thought that the learning approach fits their learning preference and promote their interest in learning software security. They expressed their thoughts about the advantages of the proposed learning approach. For example:

"I highly recommend your method. Teaching practice first. Developers can derive an understanding of the theory easier from the practice instead of doing it the other way round."

"Software security needs to be practical; it needs to be related to something, to be given the contrast to something. So it becomes really interesting when I reach your ideas. But where there is so much theory it's also a bit hard to understand."

Last, 71% of respondents agreed that the learning approach helped them relate security knowledge to their prior experience. One respondent supporting the statement commented:

"When I relate the cases to the practical things that I do in my project, the security concepts become more applicable and easier to understand."

However, we found that the survey item, Experience correlation, got the least satisfaction (Mean = 4.05) in the category. Seven respondents, that is one-third, did not hold a positive agreement with the statement, and the neutral responses were

Table 15.4: Descriptive analysis of the proposed learning approach

	Frequency						
Item	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Mean	Std. Deviation
Difficulty reduction	0	0	3	11	7	4.19	0.700
Effectiveness	0	0	2	14	5	4.14	0.573
Learning preference	0	0	4	10	7	4.14	0.973
Interest promotion	0	0	4	8	9	4.24	0.949
Experience correlation	0	1	5	7	8	4.05	0.928

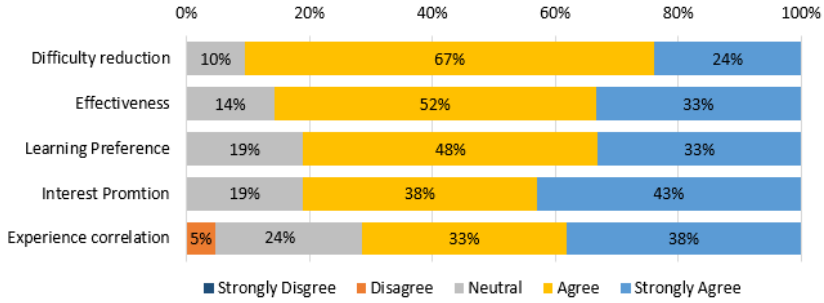


Figure 15.9: Stacked bar chart: responses to questions of the proposed learning approach based on 5-point Likert scale

relatively high (six respondents). Probing into this issue, we identified respondents' comments related to this survey item. They reported that their specialties were not within the knowledge scope that the system currently provided. For example, a respondent who was familiar with Python stated:

" I've used Python for many years. I expect this [programming language] will be included in your code examples."

15.7 Discussion

The results of this study indicate that our proposed learning system has the potential to be an effective learning tool that can motivate OSS developers to learn about software security. First, the respondents overall evaluated the practicality of system features with a positive degree. They highly recommended the use of software scenarios with graphical and contextualized security knowledge presentation. With a clear and visualized layout, they could sort out the desired knowledge quickly. Second, the results also indicated the learning approach kept developers interested and engaged. They overwhelmingly expressed their satisfaction with the learning sessions. Such benefits of the contextualized approach can be explained by the effective mechanism of intrinsic motivation, where a learner is drawn to engage in a task because it is perceived as interesting, enjoyable, and/or useful [89, 115, 251].

Based on the findings presented in the study, we deem contextualized learning a suitable approach to support developers' security training and education in software projects. In OSS, development, and maintenance of qualified and secured software products rely mainly on the ability of participants to acquire, refine and use new aspects of secure programming knowledge in their projects [483]. With proper contextual guidance, developers can identify their necessary security information, improve learning efficiency and make security knowledge more meaningful for their software development tasks. The contextualized approach helped the developers to see how the various security concepts were inter-related in their works and gave them the personalized perspective that they valued. Therefore, their learning experience can be related to a similar programming topic that they want to learn about or a problem to be solved in their projects. In addition, when developers encounter the security problems within the context they are already familiar with, the consequences of exploiting the code's vulnerabilities will be understood with a strong and personal effect, which becomes more real and less theoretical.

From this study, we also draw some lessons for further improvements to this learning system. First, we need to create more contextual scenarios and equip corresponding security knowledge in the system to expand the knowledge scope. The learning sessions can then be cast in the contexts, which are more closed to learners' working environments. Additionally, the respondents also indicated that they could not grasp the abstract explanation of security concepts because of the heavy embedded textual descriptions. The abstraction knowledge we built was extracted from the resources on the internet (e.g. OWASP and CWE). It is suggested that we decompose the vast information into smaller knowledge objects to further ease learners' loading. With the defined relationships in the ontology, these new instances can also be illustrated in the concept map to support knowledge navigation. For example, the security practice of "Input validation" can be broken down into flat text validation, rich text validation, and file upload validation, etc. We are proactively working on this improvement in preparing for longer-term studies.

15.8 Conclusion

In this study, a web-based learning system was conceptualized and developed to support contextualized learning about software security. We have presented the design rationale, including the embedded learning strategies and underlying ontological knowledge repository. Our approach attempts to place security learning in the context of software projects that can draw developers' attention to similar software events and conditions. We aim to help learners organize security knowledge by connecting concepts to real software scenarios, to motivate learners and stimulate their interest. The contextualization of security knowledge makes it possible to support developers to reflect on their learning to bridge ideas from a familiar concrete context so they can recognize their own personal relationship to these concepts.

The proposed learning system was evaluated through an online survey with 21 developers in OSS projects. Overall, the analysis of the survey data yielded positive and promising results, in which OSS participants overwhelmingly expressed their satisfaction with our proposal, in perspectives of system features and the embedded learning approach. They enjoyed the experience, found the subject matter interesting and found the presentation helpful. This finding demonstrates that our approach is not only possible but also practical to be adopted by software development projects. We are encouraged by the results of the context-based approach and believe it provides a formula for increasing the attitude and understanding of security subjects for developers without sacrificing rigor or quality of learning. We believe this implies a direct effect of the contextualized learning approach on higher overall learning satisfaction, which motivates developers to learn.

Several limitations of this study should be noted. First, this evaluation was based on self-reported data from voluntary participants about their experience and perceptions of the proposed learning system. It is not certain their actual behavior on the system, the span of time they practice the system, and for how long the knowledge will be retained. Moreover, the number of respondents obtained from the survey was relatively small compared with the enormous number of OSS projects and field workers today. We intend to invite more OSS participants from various domains joining future sessions, meanwhile, to conduct in-depth interviews to collect more detailed information about their thoughts and learning behaviors.

Bibliography

[1] Abunadi, I. and M. Alenezi (2015), "Towards cross project vulnerability prediction in open source web applications". in Proceedings of the The International Conference on Engineering & MIS 2015. ACM.

[2] Acar, Y., S. Fahl, and M.L. Mazurek (2016), "You are not your developer, either: A research agenda for usable security and privacy research beyond end users". in 2016 IEEE Cybersecurity Development (SecDev). IEEE.

[3] Ackerman, M.S. (1996), "Definitional and contextual issues in organizational and group memories". Information Technology & People, volume 9, issue 1, pages 10-24.

[4] Agarwal, R. and J. Prasad (1999), "Are individual differences germane to the acceptance of new information technologies?". Decision sciences, volume 30, issue 2, pages 361-391.

[5] Ajith Kumar, J. and L. Ganesh (2009), "Research on knowledge transfer in organizations: a morphology". Journal of knowledge management, volume 13, issue 4, pages 161-174.

[6] Ajzen, I., M.J.J.o.p. Fishbein, and S. Psychology (1973), "Attitudinal and normative variables as predictors of specific behavior". volume 27, issue 1, pages 41.

[7] Al Sabbagh, B. and S. Kowalski (2012), "Developing social metrics for security modeling the security culture of it workers individuals (case study)". in Communications, Computers and Applications (MIC-CCA), 2012 Mosharaka International Conference on. IEEE.

[8] Al Sabbagh, B. and S. Kowalski (2013), "A socio-technical framework for threat modeling a software supply chain". in The 2013 Dewald Roode Workshop on Information Systems Security Research, October 4-5, 2013, Niagara Falls, New York, USA. International Federation for Information Processing.

[9] Alavi, M. and D. Leidner (1999), "Knowledge management systems: issues, challenges, and benefits". Communications of the Association for Information systems, volume 1, issue 1, pages 7.

[10] Alenezi, M. and Y. Javed (2016), "Open source web application security: A static analysis approach". in Engineering & MIS (ICEMIS), International Conference on. IEEE.

[11] Alexander Hars, S.O. (2002), "Working for free? Motivations for participating in open-source projects". International Journal of Electronic Commerce, volume 6, issue 3, pages 25-39.

[12] Alhababi, H.H. (2017), "Technological Pedagogical Content Knowledge (Tpack) Effectiveness on English Teachers And Students in Saudi Arabia".

[13] Alnaeli, S.M., et al. (2016), "On the evolution of mobile computing software systems and C/C++ vulnerable code: Empirical investigation". in Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), IEEE Annual. IEEE.

[14] Alqahtani, S.S., E.E. Eghan, and J. Rilling (2016), "Tracing known security vulnerabilities in software repositories–A Semantic Web enabled modeling approach". Science of Computer Programming, volume 121, issue, pages 153-175.

[15] Altinkemer, K., J. Rees, and S. Sridhar (2008), "Vulnerabilities and patches of open source software: an empirical study". Journal of Information System Security, volume 4, issue 2, pages 3-25.

[16] Anbalagan, P. and M. Vouk (2009), "Towards a unifying approach in understanding security problems". in ISSRE'09. 20th International Symposium on Software Reliability Engineering. IEEE.

[17] Anbalagan, P. and M. Vouk (2010), "Towards a bayesian approach in modeling the disclosure of unique security faults in open source projects". in IEEE 21st International Symposium on Software Reliability Engineering (ISSRE). IEEE.

[18] Anderson, R. (1997), "Work Ethnography and System Design. The Encyclopedia of MicroComputers 20, A. Kent and JG Williams". Marcel Dekker.

[19] Anderson, R.C., R.W. Kulhavy, and T. Andre (1972), "Conditions under which feedback facilitates learning from programmed lessons". Journal of Educational Psychology, volume 63, issue 3, pages 186.

[20] Anne, A. and M.A.J.C.A. Sasse (1999), "Users are not the enemy". volume 42, issue 12, pages 40-46.

[21] Anttila, J., et al. (2007), "Fulfilling the needs for information security awareness and learning in information society". in The 6th annual security conference, Las Vegas.

[22] Apple Inc., "Introduction to Secure Coding Guide"; Available from: https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html. (Accessed on November 2, 2019)

[23] Apvrille, A. and M. Pourzandi (2005), "Secure software development by example". IEEE Security & Privacy, volume 3, issue 4, pages 10-17.

[24] Argote, L. (2012), "Organizational learning: Creating, retaining and transferring knowledge". volume: Springer Science & Business Media.

[25] Au, Y.A., et al. (2009), "Virtual organizational learning in open source software development projects". Information & Management, volume 46, issue 1, pages 9-15.

[26] Avery, D., et al. (2016), "Externalization of software behavior by the mining of norms". in Proceedings of the 13th International Conference on Mining Software Repositories. ACM.

[27] Badri, M.A., D. Davis, and D. Davis (1995), "A study of measuring the critical factors of quality management". International Journal of Quality & Reliability Management, volume 12, issue 2, pages 36-53.

[28] Bakanauskienė, I. and J. Martinkienė (2011), "Determining managerial competencies of management professionals". Management of Organizations: Systematic Research volume, issue 60, pages 29-43.

[29] Bam, K. (1992), "Research methods for business and management".

[30] Banday, M.T. (2011), "Ensuring Authentication and Integrity of Open Source Software using Digital Signature". International Journal of Computer Application volume 3, issue 2, pages 11-14.

[31] Barnum, S. and G. McGraw (2005), "Knowledge for software security". IEEE Security & Privacy, volume 3, issue 2, pages 74-78.

[32] Barnum, S. and A. Sethi (2007), "Attack patterns as a knowledge resource for building secure software". in OMG Software Assurance Workshop: Cigital.

[33] Basili, V.R. and H.D. Rombach (1991), "Support for comprehensive reuse". Software engineering journal, volume 6, issue 5, pages 303-316.

[34] Baskerville, R., J. Pries-Heje, and J. Venable (2009), "Soft design science methodology". in Proceedings of the 4th international conference on design science research in information systems and technology. ACM.

[35] Bassok, M.J.C.D.i.P.S. (1996), "Using content to interpret structure: Effects on analogical transfer". volume 5, issue 2, pages 54-58.

[36] Baxter, G. and I. Sommerville (2011), "Socio-technical systems: From design methods to systems engineering". Interacting with computers, volume 23, issue 1, pages 4-17.

[37] Benbya, H. and N. Belbaly (2010), "Understanding developers' motives in open source projects: a multi-theoretical framework". volume, issue, pages.

[38] Bennett, J., F. Lubben, and S. Hogarth (2007), "Bringing science to life: A synthesis of the research evidence on the effects of context-based and STS approaches to science teaching". Science education, volume 91, issue 3, pages 347-370.

[39] Berns, R.G. and P.M. Erickson (2001), "Contextual Teaching and Learning: Preparing Students for the New Economy. ". National Dissemination Center for Career and Technical Education.

[40] Berry, L.M. and J.P. Houston (1993), "Psychology at work: An introduction to industrial and organizational psychology". Brown & Benchmark/Wm. C. Brown Publ.

[41] Bider, I., P. Johannesson, and E. Perjons (2013), "Design science research as movement between individual and generic situation-problem–solution spaces", in Designing Organizational Systems, Springer. pages 35-61.

[42] Bider, I. and S. Kowalski (2014), "A framework for synchronizing human behavior, processes and support systems using a socio-technical approach", in Enterprise, Business-Process and Information Systems Modeling, Springer. pages 109-123.

[43] Biggs, J.B. (2011), "Teaching for quality learning at university: What the student does". McGraw-hill education (UK).

[44] Birkenkrahe, M. (2002), "How large multi-nationals manage their knowledge". Business Review, volume 4, issue 2, pages 2-12.

[45] Bishop, A.J. (1988), "Mathematics education in its cultural context". Educational studies in mathematics, volume 19, issue 2, pages 179-191.

[46] Bishop, M. (2010), "A Clinic for" Secure" Programming". IEEE Security & Privacy, volume 8, issue 2, pages 54-56.

[47] Bishop, M., et al. (2017), "Evaluating secure programming knowledge". in IFIP World Conference on Information Security Education. Springer.

[48] Bishop, M. and D.A. Frincke (2005), "Teaching secure programming". IEEE Security and Privacy, volume 3, issue 5, pages 54-56.

[49] Black Duck Software, "Security in the age of open source ", https://www.slideshare.net/blackducksoftware/september-13-2016-security-in-the-age-of-open-source. (Accessed on Nov. 18, 2017)

[50] BlackDuck Software, "2017 Open Source Security and Risk Analysis", Web: https://www.blackducksoftware.com/open-source-security-risk-analysis-2017. (Accessed on Sep. 13, 2018)

[51] Boh, W.F. (2007), "Mechanisms for sharing knowledge in project-based organizations". Information and organization, volume 17, issue 1, pages 27-58.

[52] Bosu, A. (2014), "Characteristics of the vulnerable code changes identified through peer code review". in Companion Proceedings of the 36th International Conference on Software Engineering. ACM.

[53] Bosu, A. and J.C. Carver (2014), "Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation". in Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM.

[54] Bosu, A., et al. (2014), "Identifying the characteristics of vulnerable code changes: An empirical study". in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM.

[55] Bosu, A., et al. (2014), "When are OSS developers more likely to introduce vulnerable code changes? A case study". in IFIP International Conference on Open Source Systems. Springer.

[56] Brank, J., M. Grobelnik, and D. Mladenic (2005), "A survey of ontology evaluation techniques". in Proceedings of the conference on data mining and data warehouses (SiKDD 2005). Citeseer Ljubljana, Slovenia.

[57] Brézillon, P. (1999), "Context in problem solving: A survey". The Knowledge Engineering Review, volume 14, issue 1, pages 47-80.

[58] Brézillon, P. (2002), "Modeling and using context: Past, present and future". in Rapport de recherche interne LIP6. Paris.

[59] Brézillon, P. (2003), "Making context explicit in communicating objects". Communicating with Smart Objects: Developing Technology for Usable Pervasive Computing Systems, Kogan Page, London.

[60] Brézillon, P. and R. Araujo (2005), "Reinforcing shared context to improve collaboration". Revue des Sciences et Technologies de l'Information-Série RIA: Revue d'Intelligence Artificielle, volume 19, issue 3, pages 537-556.

[61] Brézillon, P. and J.-C. Pomerol (1999), "Contextual knowledge sharing and cooperation in intelligent assistant systems". Le Travail Humain, volume 62, issue 3, pages 223-246.

[62] Brooking, A. (1999), "Corporate memory: Strategies for knowledge management". volume: Cengage Learning EMEA.

[63] Brooks Jr, F.P. (1995), "The Mythical Man-Month: Essays on Software Engineering". Pearson Education India.

[64] Bulgurcu, B., H. Cavusoglu, and I.J.M.q. Benbasat (2010), "Information security policy compliance: an empirical study of rationality-based beliefs and information security awareness". volume 34, issue 3, pages 523-548.

[65] Busch, M. and M. Wirsing (2015), "An Ontology for Secure Web Applications". Int. J. Software and Informatics, volume 9, issue 2, pages 233-258.

[66] Cabrera, A., W.C. Collins, and J.F. Salgado (2006), "Determinants of individual engagement in knowledge sharing". The International Journal of Human Resource Management, volume 17, issue 2, pages 245-264.

[67] Caldwell, B.S. (2008), "Knowledge sharing and expertise coordination of event response in organizations". Applied ergonomics, volume 39, issue 4, pages 427-438.

[68] Campbell, D.T. (1957), "Factors relevant to the validity of experiments in social settings". Psychological bulletin, volume 54, issue 4, pages 297.

[69] Campion, M.A., et al. (2011), "Doing competencies well: Best practices in competency modeling". volume 64, issue 1, pages 225-262.

[70] CAPEC, "CAPEC Glossary - Attack Pattern"; Available from: https://capec.mitre.org/about/glossary.html. (Accessed on Jun. 3, 2019)

[71] Cerone, A. and S.K. Sowe (2010), "Using free/libre open source software projects as e-learning tools". in Electronic Communications of the EASST.

[72] Chandra, P., "The Software Assurance Maturity Model-A guide to building security into software development"; Available from: https://www.opensamm.org/. (Accessed on Octorber 13, 2019)

[73] Chehrazi, G., I. Heimbach, and O. Hinz (2016), "The impact of security by design on the success of open source software". in Research Papers. ECIS 2016 Proceedings, Paper 179.

[74] Chen, C.C., et al. (2008), "A cross-cultural investigation of situational information security awareness programs". volume 16, issue 4, pages 360-376.

[75] Chen, X., et al. (2013), "Knowledge sharing in open source software project teams: A transactive memory system perspective". International Journal of Information Management, volume 33, issue 3, pages 553-563.

[76] Chen, X., et al. (2016), "Mechanisms of knowledge sharing in open source software projects: a comparison of Chinese and Western practice". International Journal of Technology Intelligence and Planning, volume 11, issue 2, pages 117-139.

[77] Chen, X., et al. (2017), "Managing knowledge sharing in distributed innovation from the perspective of developers: empirical study of open source software projects in China". Technology Analysis & Strategic Management, volume 29, issue 1, pages 1-22.

[78] Cheng, C.K. and F.J. Kurfess (2003), "A Context-Based Knowledge Management Framework for Software Development". CONQUEST.

[79] Chi, M.T. (2006), "Two approaches to the study of experts' characteristics", The Cambridge handbook of expertise and expert performance. pages 21-30.

[80] Chia, P., S. Maynard, and A. Ruighaver (2002), "Understanding organizational security culture". in Proceedings of PACIS. Japan.

[81] Child, J. (1984), "Organization: A guide to problems and practice". Sage.

[82] Chung, S. and B. Endicott-Popovsky (2010), "Software reengineering based security teaching". in Proceedings of the 7th Annual International Conference on International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2010). Orlando, FL.

[83] Clarke, R., D. Dorwin, and R. Nash (2009), "Is open source software more secure?". in Homeland Security/Cyber Security.

[84] Clarke, R., D. Dorwin, and R. Nash (2009), "Is open source software more secure?". Homeland Security/Cyber Security, volume, issue, pages.

[85] Cognition Technology Group at Vanderbilt (1992), "Anchored instruction in science and mathematics: Theoretical basis, developmental projects, and initial research findings". Philosophy of science, cognitive psychology, pages 244-273.

[86] Cognition Technology Group at Vanderbilt (1992), "The Jasper series as an example of anchored instruction: Theory, program description, and assessment data". Educational Psychologist, volume 27, issue 3, pages 291-315.

[87] Colomina, I., J. Arnedo-Moreno, and R. Clarisó (2013), "A study on practices against malware in free software projects". in 2013 27th International Conference on Advanced Information Networking and Applications Workshops. IEEE.

[88] Cooper, S. and S. Cunningham (2010), "Teaching computer science in context". Acm Inroads, volume 1, issue 1, pages 5-8.

[89] Cordova, D.I. and M.R. Lepper (1996), "Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice". Journal of educational psychology, volume 88, issue 4, pages 715.

[90] Council, N.R. (2000), "How people learn: Brain, mind, experience, and school: Expanded edition". volume: National Academies Press.

[91] Council, N.R. (2003), "Evaluating and improving undergraduate teaching: In". Science, Technology, Engineering, and Mathematics, volume, issue, pages.

[92] Cowan, C. (2003), "Software security for open-source systems". IEEE Security & Privacy, volume 99, issue 1, pages 38-45.

[93] Cox, A., S. Connolly, and J.J.V. Currall (2001), "Raising information security awareness in the academic setting". volume 31, issue 2, pages 11-16.

[94] Craik, K.J.W. (1967), "The nature of explanation". volume 445. CUP Archive.

[95] Creswell, J.W. and V.L.P. Clark (2017), "Designing and conducting mixed methods research". Sage publications.

[96] Criu, R. and A. Marian (2014), "The influence of students' perception of pedagogical content knowledge on self-efficacy in self-regulating learning ". Procedia-Social and Behavioral Sciences, volume 142, issue, pages 673-678.

[97] Crowston, K. and B. Scozzi (2008), "Bug fixing practices within free/libre open source software development teams". Journal of Database Management volume 19, issue 2, pages 1-30.

[98] Crowston, K., et al. (2012), "Free/Libre open-source software development: What we know and what we do not know". ACM Computing Surveys (CSUR), volume 44, issue 2, pages 7.

[99] Cummings, J.N., J.A. Espinosa, and C.K. Pickering (2009), "Crossing spatial and temporal boundaries in globally distributed projects: A relational model of coordination delay". Information Systems Research, volume 20, issue 3, pages 420-439.

[100] Curtis, B., H. Krasner, and N. Iscoe (1988), "A field study of the software design process for large systems". Communications of the ACM, volume 31, issue 11, pages 1268-1287.

[101] Cusumano, M.A. and R.W. Selby (1998), "Microsoft secrets: how the world's most powerful software company creates technology, shapes markets, and manages people". volume: Simon and Schuster.

[102] CVE, "Browse Vulnerabilities By Date"; Available from: https://www.cvedetails.com/browse-by-date.php. (Accessed on May 3, 2019)

[103] CWE, "CWE Glossary-Vulnerability"; Available from: https://cwe.mitre.org/documents/glossary/index.html#Vulnerability. (Accessed on Jun. 3, 2019)

[104] CWE, "CWE Glossary-Weakness"; Available from: https://cwe.mitre.org/documents/glossary/index.html#Weakness. (Accessed on June 3, 2019)

[105] Da Veiga, A. and J.H. Eloff (2010), "A framework and assessment instrument for information security culture". Computers & Security, volume 29, issue 2, pages 196-207.

[106] Dabbish, L., et al. (2012), "Social coding in GitHub: transparency and collaboration in an open software repository". in Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work. ACM.

[107] Dallas, S. and M. Bell (2004), "The need for IT governance: Now more than ever". Gartner Inc.

[108] Damaševičius, R. (2007), "Analysis of software design artifacts for socio-technical aspects". INFOCOMP Journal of Computer Science, volume 6, issue 4, pages 7-16.

[109] Damaševičius, R. (2009), "On the human, organizational, and technical aspects of software development and analysis", in Information Systems Development, Springer. pages 11-19.

[110] Damiani, E., C.A. Ardagna, and N. El Ioini (2009), "OSS security certification", in Open Source Systems Security Certification, Springer. pages 1-36.

[111] Dark, M., et al. (2015), "Teach the hands, train the mind... a secure programming clinic".

[112] David, W. and L. Fahey (2000), "Diagnosing cultural barriers to knowledge management". The Academy of management executive, volume 14, issue 4, pages 113-127.

[113] Davtyan, R. (2014), "Contextual learning". in ASEE 2014 Zo. 1 Conf.

[114] De Souza, C., et al. (2004), "From technical dependencies to social dependencies". in Workshop on Social Networks for Design and Analysis: Using Network Information in CSCW.

[115] Dean, R.J. and L. Dagostino (2007), "Motivational factors affecting advanced literacy learning of community college students". Community College Journal of Research Practice, volume 31, issue 2, pages 149-161.

[116] DeVito, J.A. (2002), "Human communication". Boston: Allyn & Bacon.

[117] Dey, A.K. (2001), "Understanding and using context". Personal ubiquitous computing, volume 5, issue 1, pages 4-7.

[118] Dhillon, G. (1997), "Managing information system security". Macmillan International Higher Education.

[119] Dhillon, G. (2001), "Challenges in managing information security in the new millennium", in Information security management: Global challenges in the new millennium, IGI Global. pages 1-8.

[120] Diethelm, I., P. Hubwieser, and R. Klaus (2012), "Students, teachers and phenomena: educational reconstruction for computer science education". in Proceedings of the 12th Koli Calling International Conference on Computing Education Research. ACM.

[121] DiPietro, M., et al. (2010), "Best practices in teaching K-12 online: Lessons learned from Michigan Virtual School teachers". Journal of interactive online learning, volume 9, issue 3, pages 10.

[122] Dolmans, D.H., et al. (2005), "Problem-based learning: Future challenges for educational practice and research". Medical education, volume 39, issue 7, pages 732-741.

[123] Ducheneaut, N. (2005), "Socialization in an open source software community: A socio-technical analysis". Computer Supported Cooperative Work (CSCW), volume 14, issue 4, pages 323-368.

[124] Duhon, H.J. and J. Elias (2007), "Why It's Difficult To Learn Lessons: Insights from Decision Theory and Cognitive Science". in SPE Annual Technical Conference and Exhibition. Society of Petroleum Engineers.

[125] Easterbrook, S., et al. (2008), "Selecting empirical methods for software engineering research", in Guide to advanced empirical software engineering, Springer. pages 285-311.

[126] Edwards, N. and L. Chen (2012), "An historical examination of open source releases and their vulnerabilities". in Proceedings of the 2012 ACM conference on Computer and communications security. ACM.

[127] Eisenhardt, K.M. and M.E. Graebner (2007), "Theory building from cases: Opportunities and challenges". Academy of management journal, volume 50, issue 1, pages 25-32.

[128] Emery, F.E. (1959), "Characteristics of socio-technical systems: A critical review of theories and facts about the effects of technological change on the internal structure of work organisations; with special reference to the effects of higher mechanisation and automation". volume: Tavistock Institute of Human Relations.

[129] Ericsson, K.A. and A.C. Lehmann (1996), "Expert and exceptional performance: Evidence of maximal adaptation to task constraints". Annual review of psychology, volume 47, issue 1, pages 273-305.

[130] Errington, E.P.J.I.J.o.L. (2009), "Being there: closing the gap between learners sand contextual knowledge using near-world scenarios". volume 16, issue, pages 585-594.

[131] Erturk, E. (2012), "A case study in open source software security and privacy: Android adware". in World Congress on Internet Security (WorldCIS-2012). IEEE.

[132] Eskerod, P. and H.J.g. Skriver (2007), "Organizational culture restraining in-house knowledge transfer between project managers--a case study". Project Management Institute.

[133] Espinosa, J.A., et al. (2007), "Team knowledge and coordination in geographically distributed software development". Journal of management information systems, volume 24, issue 1, pages 135-169.

[134] Faraj, S. and L. Sproull (2000), "Coordinating expertise in software development teams". Management science, volume 46, issue 12, pages 1554-1568.

[135] Farrior, M. (2005), "Breakthrough strategies for engaging the public: Emerging trends in communications and social science". Biodiversity Project. February

[136] Felder, R.M. and L.K.J.E.e. Silverman (1988), "Learning and teaching styles in engineering education". volume 78, issue 7, pages 674-681.

[137] Felder, R.M., et al. (2000), "The future of engineering education II. Teaching methods that work". volume 34, issue 1, pages 26-39.

[138] Feller, J., et al. (2006), "Developing open source software: a community-based analysis of research", in Social Inclusion: Societal and Organizational Implications for Information Systems, Springer. pages 261-278.

[139] Feller, J. and B. Fitzgerald (2000), "A framework analysis of the open source software development paradigm". in Proceedings of the twenty first international conference on Information systems. Association for Information Systems.

[140] Feller, J. and B. Fitzgerald (2002), "Understanding open source software development". Addison-Wesley London.

[141] Feng, Q., et al. (2016), "Towards an architecture-centric approach to security analysis". in 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE.

[142] Fenstermacher, G.D. and V. Richardson (2005), "On making determinations of quality in teaching". Teachers college record, volume 107, issue 1, pages 186-213.

[143] Fenz, S. and A. Ekelhart (2009), "Formalizing information security knowledge". in Proceedings of the 4th international Symposium on information, Computer, and Communications Security. ACM.

[144] Fernandes, S., et al. (2013), "Integrating formal and informal learning through a FLOSS-based innovative approach". in International Conference on Collaboration and Technology. Springer.

[145] Ferrario, M.A., et al. (2014), "Software engineering for'social good': integrating action research, participatory design, and agile development". in Companion Proceedings of the 36th International Conference on Software Engineering. ACM.

[146] Fetters, M.D., L.A. Curry, and J.W. Creswell (2013), "Achieving integration in mixed methods designs—principles and practices". Health services research, volume 48, issue 6pt2, pages 2134-2156.

[147] Fischer, C. and S. Gregor (2011), "Forms of reasoning in the design science research process". in International Conference on Design Science Research in Information Systems. Springer.

[148] Fong Boh, W., S.A. Slaughter, and J.A. Espinosa (2007), "Learning from experience in software development: A multilevel analysis". Management science, volume 53, issue 8, pages 1315-1331.

[149] Fonseca, J. and M. Vieira (2013), "A survey on secure software development lifecycles", in Software Development Techniques for Constructive Information Systems Design, IGI Global. pages 57-73.

[150] Ford, D.P. and Y.E. Chan (2003), "Knowledge sharing in a multi-cultural setting: a case study". Knowledge Management Research & Practice, volume 1, issue 1, pages 11-27.

[151] Fortify's Security Research Group (2008), "Open Source Security Study: How Are Open Source development communities embracing Security Best practices?".

[152] Fox, W.M. (1995), "Sociotechnical system principles and guidelines: past and present". The Journal of Applied Behavioral Science, volume 31, issue 1, pages 91-105.

[153] Fulcher, A. and P. Hills (1996), "Towards a strategic framework for design research". Journal of Engeering Design, volume 7, issue 2, pages 183-193.

[154] Furnell, S.J.C. and Security (2007), "From the Editor-in-Chief: IFIP workshop-Information security culture". volume 26, issue 1, pages 35.

[155] Futcher, L. and R. von Solms (2008), "Guidelines for secure software development". in Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology.

[156] Gabel, D. (1999), "Improving teaching and learning through chemistry education research: A look to the future". Journal of Chemical education, volume 76, issue 4, pages 548.

[157] Garcia, J., et al. (2009), "Toward a catalogue of architectural bad smells". in International Conference on the Quality of Software Architectures. Springer.

[158] Garrison, D.R. (1997), "Self-directed learning: Toward a comprehensive model". Adult education quarterly, volume 48, issue 1, pages 18-33.

[159] Gasser, L., et al. (2003), "Understanding continuous design in F/OSS projects". in In 16th. Intern. Conf. Software & Systems Engineering and their Applications. Citeseer.

[160] Gentner, D. and A.L. Stevens (2014), "Mental models". volume: Psychology Press.

[161] Giamellaro, M.J.I.J.o.S.E. (2014), "Primary contextualization of science learning through immersion in content-rich settings". volume 36, issue 17, pages 2848-2871.

[162] Gilbert, J.K. (2006), "On the nature of "context" in chemical education". International journal of science education, volume 28, issue 9, pages 957-976.

[163] GitHub, "Celebrating nine years of GitHub with an anniversary sale"; Available from: https://github.com/blog/2345-celebrating-nine-years-of-github-with-an-anniversary-sale. (Accessed on March 3, 2019)

[164] GitHub, "Github user search"; Available from: https://github.com/search?q=type:user&type=Users. (Accessed on March 3, 2019)

[165] Glass, R., V. Ramesh, and I. Vessey (2004), "An analysis of research in computing disciplines".

[166] Glass, R.L. (2001), "Frequently forgotten fundamental facts about software engineering". IEEE software, volume, issue 3, pages 112,110-111.

[167] Glass, R.L., I. Vessey, and V. Ramesh (2002), "Research in software engineering: an analysis of the literature". Information and Software technology, volume 44, issue 8, pages 491-506.

[168] Godfrey, M.W. and Q. Tu (2000), "Evolution in open source software: A case study". in Software Maintenance, 2000. Proceedings. International Conference on. IEEE.

[169] Goldkuhl, G. and E. Braf (2001), "Contextual knowledge analysis-understanding knowledge and its relations to action and communication". in Second European Conference on Knowledge Management Proceedings.

[170] Goldstone, R.L. and Y. Sakamoto (2003), "The transfer of abstract principles governing complex adaptive systems". Cognitive psychology, volume 46, issue 4, pages 414-466.

[171] Goldstone, R.L. and J.Y. Son (2005), "The transfer of scientific principles using concrete and idealized simulations". The Journal of the Learning Sciences, volume 14, issue 1, pages 69-110.

[172] Gousios, G., et al. (2014), "Lean GHTorrent: GitHub data on demand". in Proceedings of the 11th working conference on mining software repositories. ACM.

[173] Graff, M. and K.R. Van Wyk (2003), "Secure coding: principles and practices". O'Reilly Media, Inc.

[174] Gravemeijer, K.P. (1994), "Developing realistic Mathematics Education ".

[175] Green, M. and M. Smith (2016), "Developers are not the enemy!: The need for usable security apis". IEEE Security & Privacy, volume 14, issue 5, pages 40-46.

[176] Gregor, S. and A.R. Hevner (2013), "Positioning and presenting design science research for maximum impact", MIS quarterly. pages 337-355.

[177] Grinter, R.E. (2003), "Recomposition: Coordinating a web of software dependencies". Computer Supported Cooperative Work (CSCW), volume 12, issue 3, pages 297-327.

[178] Grodzinsky, F.S., et al. (2003), "Ethical issues in open source software". volume 1, issue 4, pages 193-205.

[179] Groven, A.-K., et al. (2010), "Security measurements within the framework of quality assessment models for free/libre open source software". in Proceedings of the 4th European conference on Software Architecture. ACM.

[180] Gruber, T.R. (1993), "A translation approach to portable ontology specifications". Knowledge acquisition, volume 5, issue 2, pages 199-220.

[181] Gruber, T.R. (1995), "Toward principles for the design of ontologies used for knowledge sharing?". International journal of human-computer studies, volume 43, issue 5, pages 907-928.

[182] Guan, H., H. Yang, and J. Wang (2016), "An ontology-based approach to security pattern selection". International Journal of Automation and Computing, volume 13, issue 2, pages 168-182.

[183] Guo, M. and J.A. Wang (2009), "An ontology-based approach to model common vulnerabilities and exposures in information security". in ASEE Southest Section Conference.

[184] Guzdial, M. (2010), "Does contextualized computing education help?". ACM Inroads, volume 1, issue 4, pages 4-6.

[185] Gyrard, A., C. Bonnet, and K. Boudaoud (2013), "The stac (security toolbox: attacks & countermeasures) ontology". in Proceedings of the 22nd International Conference on World Wide Web. ACM.

[186] Haas, M.R. and M.T. Hansen (2007), "Different knowledge, different benefits: Toward a productivity perspective on knowledge sharing in organizations". Strategic Management Journal, volume 28, issue 11, pages 1133-1153.

[187] Hair, J.F., et al. (1998), "Multivariate data analysis". volume 5. Prentice hall Upper Saddle River, NJ.

[188] Hales, D. and C. Douce (2002), "Modelling Software Organisations". in Proc. of PPIG.

[189] Hanson, N.R. (1958), "The logic of discovery". The Journal of Philosophy, volume 55, issue 25, pages 1073-1089.

[190] Hardi, J. (2010), "Situated Learning among Open Source Software Developers: The Case of Google Chrome Project".

[191] Harnesk, D., J.J.I.M. Lindström, and C. Security (2011), "Shaping security behaviour through discipline and agility: Implications for information security management". volume 19, issue 4, pages 262-276.

[192] Hauge, Ø., C. Ayala, and R. Conradi (2010), "Adoption of open source software in software-intensive organizations–A systematic literature review". Information and Software Technology, volume 52, issue 11, pages 1133-1154.

[193] Hayes, J.R. (2013), "The complete problem solver". volume: Routledge.

[194] Hazeyama, A. (2012), "Survey on body of knowledge regarding software security". in Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on. IEEE.

[195] Hazeyama, A. and H. Shimizu (2011), "A learning environment for software security education". in 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement-Companion. IEEE.

[196] Hazeyama, A. and H. Shimizu (2012), "Development of a Software Security Learning Environment". in 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. IEEE.

[197] Hemetsberger, A. and C. Reinhardt (2004), "Sharing and creating knowledge in open-source communities: the case of KDE". in Paper for Fifth European Conference on Organizational Knowledge, Learning, and Capabilities, Innsbruck.

[198] Hemetsberger, A. and C. Reinhardt (2006), "Learning and knowledge-building in open-source communities: A social-experiential approach". Management learning, volume 37, issue 2, pages 187-214.

[199] Henninger, S. (1997), "Case-based knowledge management tools for software development". Automated Software Engineering, volume 4, issue 3, pages 319-340.

[200] Herbsleb, J.D. (2007), "Global software engineering: The future of socio-technical coordination". in 2007 Future of Software Engineering. IEEE Computer Society.

[201] Herrmann, T. (2003), "Learning and teaching in socio-technical environments", in Informatics and the Digital Society, Springer. pages 59-71.

[202] Hevner, A. and S. Chatterjee (2010), "Design research in information systems: theory and practice". volume 22. Springer Science & Business Media.

[203] Hevner, A.R., et al. (2004), "Design science in information systems research". MIS quarterly, volume 28, issue 1, pages 75-105.

[204] Hippel, E.v. and G.v. Krogh (2003), "Open source software and the "private-collective" innovation model: Issues for organization science". Organization science, volume 14, issue 2, pages 209-223.

[205] Hlomani, H. and D.J.S.W.J. Stacey (2014), "Approaches, methods, metrics, measures, and subjectivity in ontology evaluation: A survey". volume 1, issue 5, pages 1-11.

[206] Hoglund, G. and G. McGraw (2004), "Exploiting software: How to break code". Pearson Education India.

[207] Holdt Christensen, P. (2007), "Knowledge sharing: moving away from the obsession with best practices". Journal of knowledge management, volume 11, issue 1, pages 36-47.

[208] Hoq, K.M.G. (2014), "Information overload: Causes, consequences and remedies-A study". Philosophy and progress, pages 49-68.

[209] Howard, M. (2004), "Building more secure software with improved development processes". IEEE Security & Privacy, volume 2, issue 6, pages 63-65.

[210] Howard, M. and D. LeBlanc (2003), "Writing secure code". Pearson Education.

[211] Howard, M. and S. Lipner (2006), "The security development lifecycle". volume 8. Microsoft Press Redmond.

[212] Humes, L.L. (2007), "Communities of Practice for Open Source Software", in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, IGI Global. pages 610-623.

[213] Humphrey, W.S. (1995), "A discipline for software engineering". volume: Addison-Wesley Longman Publishing Co., Inc.

[214] Iivari, J. (2007), "A paradigmatic analysis of information systems as a design science". Scandinavian journal of information systems, volume 19, issue 2, pages 5.

[215] Inkpen, A.C. and E.W. Tsang (2005), "Social capital, networks, and knowledge transfer". Academy of management review, volume 30, issue 1, pages 146-165.

[216] Irvine, C.E., M.F. Thompson, and K. Allen (2005), "CyberCIEGE: gaming for information assurance". IEEE Security & Privacy, volume 3, issue 3, pages 61-64.

[217] Iskoujina, Z. and J. Roberts (2015), "Knowledge sharing in open source software communities: motivations and management". Journal of Knowledge Management, volume 19, issue 4, pages 791-813.

[218] Jaatun, M.G., et al. (2011), "A Lightweight Approach to Secure Software Engineering", A Multidisciplinary Introduction to Information Security. pages 183.

[219] Jafari, M., et al. (2008), "Exploring the contextual dimensions of organization from knowledge management perspective". VINE, volume 38, issue 1, pages 53-71.

[220] Jarzombek, J. and K.M. Goertzel (2006), "Security in the Software Lifecycle". CrossTalk: The Journal of Defense Software Engineering, volume, issue, pages.

[221] Jenkins III, C.C., T. Kitchel, and B. Hains (2010), "Defining agricultural education instructional quality". Journal of Agricultural Education, volume 51, issue 3, pages 53-63.

[222] Jiang, J.J., G. Klein, and H.-G. Chen (2006), "The effects of user partnering and user non-support on project performance". Journal of the Association for Information Systems, volume 7, issue 1, pages 6.

[223] Johannesson, P. and E. Perjons (2014), "An introduction to design science". Springer.

[224] Jonassen, D. and S. Land (2012), "Theoretical foundations of learning environments". Routledge.

[225] Jordan, T.B., et al. (2014), "Designing Interventions to Persuade Software Developers to Adopt Security Tools". in Proceedings of the 2014 ACM Workshop on Security Information Workers. ACM.

[226] Kamina, P. and N.N. Iyer (2009), "From concrete to abstract: Teaching for transfer of learning when using manipulatives". in Proceedings of the Northeastern Educational Research Association (NERA) 2009.6.

[227] Kang, W. and Y. Liang (2013), "A Security Ontology with MDA for Software Development". in Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on. IEEE.

[228] Karokola, G., L. Yngström, and S. Kowalski (2012), "Secure e-government services: A comparative analysis of e-government maturity models for the developing regions– The need for security services". International Journal of Electronic Government Research (IJEGR), volume 8, issue 1, pages 1-25.

[229] Kayworth, T. and D. Whitten (2012), "Effective information security requires a balance of social and technology factors".

[230] Keele, S. (2007), "Guidelines for performing systematic literature reviews in software engineering", Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

[231] Kern, C., A. Kesavan, and N. Daswani (2007), "Foundations of security: what every programmer needs to know". volume: Springer.

[232] Khairkar, A.D., D.D. Kshirsagar, and S. Kumar (2013), "Ontology for detection of web attacks". in Communication Systems and Network Technologies (CSNT), 2013 International Conference on. IEEE.

[233] Khan, B., et al. (2011), "Effectiveness of information security awareness methods based on psychological theories". volume 5, issue 26, pages 10862-10868.

[234] Khan, M.U.A. and M. Zulkernine (2008), "Quantifying security in secure software development phases". in 2008 32nd Annual IEEE International Computer Software and Applications Conference. IEEE.

[235] Kieras, D.E. and S.J.C.s. Bovair (1984), "The role of a mental model in learning to operate a device". volume 8, issue 3, pages 255-273.

[236] Kim, B., et al. (2015), "Design of Exploitable Automatic Verification System for Secure Open Source Software", in Advances in Computer Science and Ubiquitous Computing, Springer. pages 275-281.

[237] Kirsch, L.J.J.O.S. (1996), "The management of complex tasks in organizations: Controlling the systems development process". volume 7, issue 1, pages 1-21.

[238] Kissel, R.L., et al. (2008), "Security considerations in the system development life cycle". pages.

[239] Kitchenham, B. (2004), "Procedures for performing systematic reviews". Keele, UK, Keele University, volume 33, issue 2004, pages 1-26.

[240] Kitchenham, B. (2007), "Guidelines for performing systematic literature reviews in software engineering". Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

[241] Klein, H.K. and M.D. Myers (1999), "A set of principles for conducting and evaluating interpretive field studies in information systems". MIS quarterly, volume 23, issue 1, pages 67-94.

[242] Klemke, R. (2000), "Context Framework - an Open Approach to Enhance Organisational Memory Systems with Context Modelling Techniques". in

Proceedings of the Third International Conference on Practical Aspects of Knowledge Management (PAKM2000), 30-31 October 2000. Basel, Switzerland.

[243] Ko, A.J. and B.A. Myers (2008), "Debugging reinvented: asking and answering why and why not questions about program behavior". in Proceedings of the 30th international conference on Software engineering. ACM.

[244] Koeppen, K., et al. (2008), "Current issues in competence modeling and assessment". volume 216, issue 2, pages 61-73.

[245] Kogut, B. and A. Metiu (2001), "Open-source software development and distributed innovation". Oxford review of economic policy, volume 17, issue 2, pages 248-264.

[246] Koh, K., et al. (2005), "Security Governance: Its Impact on Security Culture". in AISM.

[247] Kolb, D. (1984), "Experiential learning as the science of learning and development". Englewood Cliffs, NJ: Prentice Hall.

[248] Koskosas, I. (2011), "Web Banking: A Security Management and Communications Approach". International Journal of Computer Science & Engineering Technology volume 2, issue 7, pages 146-154.

[249] Koulopoulos, T.M. and C. Frappaolo (1999), "Smart things to know about knowledge management". Capstone US.

[250] Kowalski, S. (1994), "IT insecurity: a multi-discipline inquiry". Department of Computer and System Sciences, University of Stockholm and Royal Institute of Technology, Sweden. .

[251] Kozeracki, C.A. (2005), "Preparing faculty to meet the needs of developmental students". New directions for community colleges, volume 129: Responding to thechallenges of developmental education, issue, pages 39-49.

[252] Kraut, R.E. and L.A. Streeter (1995), "Coordination in software development". Communications of the ACM, volume 38, issue 3, pages 69-82.

[253] Krishnamurthy, S. and A.K. Tripathi (2006), "Bounty programs in free/libre/open source software". in BITZER Jurgen, The Economics of Open Source Software Development, Lavoisier, Paris.

[254] Kruger, H.A., W.D.J.c. Kearney, and security (2008), "Consensus ranking–An ICT security awareness case study". volume 27, issue 7-8, pages 254-259.

[255] Kuechler, W., V.K. Vaishnavi, and S. Petter (2007), "The aggregate general design cycle as a perspective on the evolution of computing communities of interest", in Design Science Research Methods and Patterns, Auerbach Publications. pages 43-51.

[256] Kuhn, D.R., M. Raunak, and R. Kacker (2017), "An Analysis of Vulnerability Trends, 2008-2016". in IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE.

[257] Lakhani, K.R. and E. Von Hippel (2003), "How open source software works:“free” user-to-user assistance". Research policy, volume 32, issue 6, pages 923-943.

[258] Lakhani, K.R. and R.G. Wolf (2003), "Why hackers do what they do: Understanding motivation and effort in free/open source software projects".

[259] Land, S.M. (2000), "Cognitive requirements for learning with open-ended learning environments". Educational Technology Research and Development, volume 48, issue 3, pages 61-78.

[260] Lanzara, G.F. and M. Morner (2003), "The knowledge ecology of open-source software projects". in 19th EGOS Colloquium, Copenhagen.

[261] Larabel, M., "The Linux Kernel Has Grown By 225k Lines of Code So Far This Year From 3.3k Developers"; Available from:

https://www.phoronix.com/scan.php?page=news_item&px=Linux-September-2018-Stats. (Accessed on November 2, 2019)

[262] Lavallée, M. and P.N. Robillard (2015), "Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality". in Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press.

[263] Lave, J. (1991), "Situating learning in communities of practice". Perspectives on socially shared cognition, volume 2, issue, pages 63-82.

[264] Lave, J. and E. Wenger (1991), "Situated learning: Legitimate peripheral participation". volume: Cambridge university press.

[265] Lavrac, N. and S. Dzeroski (1994), "Inductive Logic Programming". in WLP. Springer.

[266] Leach, J., P.J.S. Scott, and Education (2003), "Individual and sociocultural views of learning in science education". volume 12, issue 1, pages 91-113.

[267] Lee, C.K. and S. Al-Hawamdeh (2002), "Factors impacting knowledge sharing". Journal of Information & Knowledge Management, volume 1, issue 01, pages 49-56.

[268] Lee Endres, M., et al. (2007), "Tacit knowledge sharing, self-efficacy theory, and application to the Open Source community". Journal of knowledge management, volume 11, issue 3, pages 92-103.

[269] Lee, J.S., J. Pries-Heje, and R. Baskerville (2011), "Theorizing in design science research". in International conference on design science research in information systems. Springer.

[270] Lee, Y.-J. and S. Chue (2013), "The value of fidelity of implementation criteria to evaluate school-based science curriculum innovations". International Journal of Science Education, volume 35, issue 15, pages 2508-2537.

[271] Lethbridge, T.C., S.E. Sim, and J. Singer (2005), "Studying software engineers: Data collection techniques for software field studies". Empirical software engineering, volume 10, issue 3, pages 311-341.

[272] Levy, J., "Top Open Source Security Vulnerabilities", WhiteSource Blog. ; Available from: https://www.whitesourcesoftware.com/whitesource-blog/open-source-security-vulnerability/. (Accessed on Feb. 22, 2017)

[273] Lewis, J. (2018), "Economic Impact of Cybercrime, No Slowing Down". volume: McAfee.

[274] Li, Z., et al. (2006), "Have things changed now?: an empirical study of bug characteristics in modern open source software". in Proceedings of the 1st workshop on Architectural and system support for improving software dependability. ACM.

[275] Lindvall, M. and I. Rus (2000), "Process diversity in software development". IEEE software, volume 17, issue 4, pages 14-18.

[276] Locke, E.A. (2007), "The case for inductive theory building". Journal of Management, volume 33, issue 6, pages 867-890.

[277] Long, S. (2013), "Socioanalytic methods: discovering the hidden in organisations and social systems". Karnac Books.

[278] Lu, S.C. and N. Jing (2009), "A socio-technical negotiation approach for collaborative design in software engineering". International Journal of Collaborative Engineering, volume 1, issue 1-2, pages 185-209.

[279] MacKenzie, D. and J. Wajcman (1999), "The social shaping of technology". volume: Open university press.

[280] Madey, G., V. Freeh, and R. Tynan (2002), "The open source software development phenomenon: An analysis based on social network theory". in AMCIS 2002 Proceedings.

[281] Mahler, J. (1997), "Influences of organizational culture on learning in public agencies". Journal of Public Administration Research and Theory, volume 7, issue 4, pages 519-540.

[282] Manzoor, S., et al. (2018), "Threat Modeling the Cloud: An Ontology Based Approach". in International Workshop on Information and Operational Technology Security Systems. Springer.

[283] March, S.T. and G.F. Smith (1995), "Design and natural science research on information technology". Decision support systems, volume 15, issue 4, pages 251-266.

[284] March, S.T. and V.C. Storey (2008), "Design science in the information systems discipline: an introduction to the special issue on design science research". MIS quarterly, volume 32, issue 4, pages 725-730.

[285] Marks, D.B. (2016), "Theory to practice: Quality instruction in online learning environments". in Society for Information Technology & Teacher Education International Conference. Association for the Advancement of Computing in Education (AACE).

[286] Markus, M.L., A. Majchrzak, and L. Gasser (2002), "A design theory for systems that support emergent knowledge processes". MIS quarterly, pages 179-212.

[287] Marques, M. and C.G. Ralha (2014), "An ontological approach to mitigate risk in web applications". in Proceedings of SBSeg 2014.

[288] Martins, A. and J. Elofe (2002), "Information security culture", in Security in the information society, Springer. pages 203-214.

[289] Martins, N., A. Da Veiga, and J.H.J.S.A.B.R. Eloff (2007), "Information security culture-validation of an assessment instrument". volume 11, issue 1, pages 147-166.

[290] May, T. (2011), "Social research". volume: Buckingham: Open University Press.

[291] McCaulley, M.H. (1976), "Psychological Types in Engineering: Implications for Teaching". Engineering Education, volume 66, issue 7, pages 729-736.

[292] McCaulley, M.H., et al. (1983), "Applications of Psychological type in engineering-education ". volume 73, issue 5, pages 394-400.

[293] McGraw, G. (2004), "Software security". IEEE Security & Privacy, volume 2, issue 2, pages 80-83.

[294] McGraw, G. (2006), "Software security: building security in". volume 1. MA,USA: Addison-Wesley Professional.

[295] McGraw, G. (2013), "Cyber war is inevitable (unless we build security in)". Journal of Strategic Studies, volume 36, issue 1, pages 109-119.

[296] McGraw, G., B. Chess, and S. Migues (2009), "Building security in maturity model". Fortify & Cigital.

[297] Mckeen, J.D., M.H. Zack, and S. Singh (2006), "Knowledge management and organizational performance: An exploratory survey". in System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on. IEEE.

[298] Mead, N.R., et al. (2004), "Software security engineering: a guide for project managers". volume: Addison-Wesley Professional.

[299] Meneely, A., et al. (2014), "An empirical investigation of socio-technical code review metrics and security vulnerabilities". in Proceedings of the 6th International Workshop on Social Software Engineering. ACM.

[300] Meneely, A. and L. Williams (2009), "Secure open source collaboration: an empirical study of linus' law". in Proceedings of the 16th ACM conference on Computer and communications security. ACM.

[301] Meneely, A. and L. Williams (2010), "Strengthening the empirical analysis of the relationship between Linus' Law and software security". in Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM.

[302] Meneely, A. and L. Williams (2011), "Socio-technical developer networks: Should we trust our measurements?". in Proceedings of the 33rd International Conference on Software Engineering. ACM.

[303] Merhi, M.I. and V. Midha (2012), "The impact of training and social norms on information security compliance: A pilot study".

[304] Merrill, M.D. (2000), "Knowledge objects and mental models". in Proceedings International Workshop on Advanced Learning Technologies (IWALT 2000) Palmerston North, New Zealand: IEEE.

[305] Meyer, D.K. and J.C. Turner (2006), "Re-conceptualizing emotion and motivation to learn ". Educational Psychology Review, volume 18, issue 4, pages 377-390.

[306] Michlmayr, M., F. Hunt, and D. Probert (2005), "Quality practices and problems in free software projects". in Proceedings of the First International Conference on Open Source Systems.

[307] Millar, R. (1998), "Beyond 2000: Science education for the future". in London: Nuffield Foundation.

[308] Mirakhorli, M., "Common architecture weakness enumeration "; Available from: http://blog.ieeesoftware.org/2016/04/common-architecture-weakness.html. (Accessed on May. 3, 2019)

[309] Mirbel, I. (2009), "OFLOSSC, an Ontology for Supporting Open Source Development Communities", ICEIS. pages 47-52.

[310] MISRA, "Guidelines for the use of the C language in critical systems"; Available from: http://caxapa.ru/thumbs/468328/misra-c-2004.pdf. (Accessed on November 2, 2019)

[311] Mockus, A., R.T. Fielding, and J.D. Herbsleb (2002), "Two case studies of open source software development: Apache and Mozilla". ACM Transactions on Software Engineering and Methodology (TOSEM), volume 11, issue 3, pages 309-346.

[312] Moha, N., et al. (2005), "A taxonomy and a first study of design pattern defects". in STEP 2005.

[313] Mohammed, N.M., et al. (2017), "Exploring software security approaches in software development lifecycle: A systematic mapping study". volume 50, issue, pages 107-115.

[314] Mourad, A., M.-A. Laverdière, and M. Debbabi (2006), "Security hardening of open source software". in Conference on Privacy, Security and Trust.

[315] Mouratidis, H. and P. Giorgini (2007), "Integrating Security and Software Engineering". in Idea Group Inc, Hershey, PA, USA.

[316] Mozilla, "WebAppSec/Secure Coding Guidelines"; Available from: https://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines. (Accessed on November 2, 2019)

[317] Murayama, K., et al. (2016), "When enough is not enough: Information overload and metacognitive decisions to stop studying information". Journal of Experimental Psychology: Learning, Memory, and Cognition, volume 42, issue 6, pages 914.

[318] Nagy, C. and S. Mancoridis (2009), "Static security analysis based on input-related software faults". in CSMR'09. 13th European Conference on Software Maintenance and Reengineering. IEEE.

[319] Naidu, S. (2008), "Situated learning designs for professional development: Fundamental principles and case studies". in Fifth Pan-Commonwealth Forum on Open Learning.

[320] Nance, K. (2009), "Teach Them When They Aren't Looking: Introducing Security in CS1". IEEE Security & Privacy, volume 7, issue 5, pages 53-55.

[321] Nance, K., B. Hay, and M. Bishop (2012), "Secure Coding Education: Are We Making Progress?".

[322] Nelson, M., R. Sen, and C. Subramaniam (2006), "Understanding open source software: A research classification framework". Communications of the Association for Information Systems, volume 17, issue 1, pages 12.

[323] Nentwig, P. and D. Waddington (2006), "Making it relevant: Context based learning of science". Waxmann Verlag.

[324] Nerbråten, Ø. and L. Røstad (2009), "Hacmegame: A tool for teaching software security". in 2009 International Conference on Availability, Reliability and Security. IEEE.

[325] Ngo, L., W. Zhou, and M. Warren (2005), "Understanding Transition towards Information Security Culture Change". in AISM.

[326] Niazi, M. (2015), "Do systematic literature reviews outperform informal literature reviews in the software engineering domain? An initial case study". Arabian Journal for Science and Engineering, volume 40, issue 3, pages 845-855.

[327] Nonaka, I. (2008), "The knowledge-creating company". volume: Harvard Business Review Press.

[328] Nonaka, I. and N. Konno (1998), "The concept of" ba": Building a foundation for knowledge creation". California management review, volume 40, issue 3, pages 40-54.

[329] Nordberg, P. (2019), "Challenges In Security Audits In Open Source Systems". pages.

[330] NorthBridge, B., "2016 Future of Open Source Survey"; Available from: http://www.northbridge.com/2016-future-open-source-survey-results. (Accessed on Nov. 23, 2017)

[331] Noy, N.F. and D.L. McGuinness (2001), "Ontology development 101: A guide to creating your first ontology".

[332] Numally, J.C. (1978), "Psychometric theory". NY: McGraw-Hill.

[333] Nunamaker Jr, J.F. and R.O. Briggs (2011), "Toward a broader vision for information systems". ACM Transactions on Management Information Systems (TMIS), volume 2, issue 4, pages 20.

[334] O'Reilly, C.A. (1982), "Variations in decision makers' use of information sources: The impact of quality and accessibility of information". Academy of Management journal, volume 25, issue 4, pages 756-771.

[335] Olchi, W.G.J.A.o.M.J. (1978), "The transmission of control through organizational hierarchy". volume 21, issue 2, pages 173-192.

[336] Oliveira, D., et al. (2014), "It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots". in Proceedings of the 30th Annual Computer Security Applications Conference. ACM.

[337] Onwuegbuzie, A.J. and J.P. Combs (2010), "Emergent data analysis techniques in mixed methods research: A synthesis". Handbook of mixed methods in social and behavioral research.

[338] Oracle Corporation, "Secure Coding Guidelines for Developers"; Available from: https://docs.oracle.com/cd/E53394_01/html/E54753/scode-1.html. (Accessed on November 2, 2019)

[339] Ouchi, W.G. (1979), "A conceptual framework for the design of organizational control mechanisms", in Readings in accounting for management control, Springer. pages 63-82.

[340] Ouchi, W.G. (1980), "Markets, bureaucracies, and clans", Administrative science quarterly. pages 129-141.

[341] OWASP, "OWASP Top 10 Application Security Risks - 2017"; Available from: https://www.owasp.org/index.php/Top_10-2017_Top_10. (Accessed on Dec. 16, 2018)

[342] OWASP, "Secure Coding Practices - Quick Reference Guide "; Available from: https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide. (Accessed on March 3, 2019)

[343] OWASP, "Comprehensive, Lightweight Application Security Process"; Available from: http://www.owasp.org,. (Accessed on October 3, 2019)

[344] OWASP, "Secure Design Principles"; Available from: https://github.com/OWASP/DevGuide/blob/master/02-Design/01-Principles%20of%20Security%20Engineering.md. (Accessed on November 2, 2019)

[345] Pan, S.L. and H. Scarbrough (1999), "Knowledge management in practice: An exploratory case study". Technology Analysis & Strategic Management, volume 11, issue 3, pages 359-374.

[346] Parchmann, I., et al. (2006), ""Chemie im Kontext": A symbiotic implementation of a context-based teaching and learning approach". volume 28, issue 9, pages 1041-1062.

[347] Park, H.S. and B.-C. Im (2003), "A Study on the Knowledge Sharing Behavior of Local Public Servants in Korea: Structural Equation Analysis". in 한국행정학회 Conference 자료.

[348] Pashler, H., et al. (2007), "Organizing Instruction and Study to Improve Student Learning. ". in National Center for Education Research. Washington, DC: Institute of Education Sciences, U.S. Department of Education.

[349] Peffers, K., et al. (2007), "A design science research methodology for information systems research". Journal of management information systems, volume 24, issue 3, pages 45-77.

[350] Peirce, C.S. (1901), "On the logic of drawing history from ancient documents, especially from testimonies". The Essential Peirce, 1893-1913, volume 2, issue, pages 75-114.

[351] Peirce, C.S. (1974), "Collected papers of charles sanders peirce". volume 2. Harvard University Press.

[352] Perin, D. (2011), "Facilitating student learning through contextualization: A review of evidence". Community College Review, volume 39, issue 3, pages 268-295.

[353] Perrone, L.F., M. Aburdene, and X. Meng (2005), "Approaches to undergraduate instruction in computer security". in Proceedings of the American Society for Engineering Education Annual Conference and Exhibition, ASEE.

[354] Pfleeger, C.P. and S.L. Pfleeger (2002), "Security in computing". volume: Prentice Hall Professional Technical Reference.

[355] Pham, R., et al. (2013), "Creating a shared understanding of testing culture on a social coding site". in 35th International Conference onSoftware Engineering (ICSE). IEEE.

[356] Piessens, F., "Cyber Security Body of Knowledge"; Available from: https://www.cybok.org/media/downloads/cybok_version_1.0.pdf. (Accessed on November 2, 2019)

[357] Pittenger, M. (2016), "Know your open source code". Network Security, volume 2016, issue 5, pages 11-15.

[358] Pohl, J. (2004), "Intelligent Software Systems for the New Infostructure".

[359] Potter, B. and G. McGraw (2004), "Software security testing". IEEE Security & Privacy, volume 2, issue 5, pages 81-85.

[360] Predmore, S.R. (2005), "Putting it into Context". Techniques: Connecting education and careers, volume 80, issue 1, pages 22-25.

[361] Purao, S. (2002), "Design research in the technology of information systems: Truth or dare", GSU Department of CIS Working Paper. pages 45-77.

[362] Race, P. (2007), "The lecturer's toolkit: A practical guide to assessment, learning and teaching". Routledge, Abingdon.

[363] Rainbird, H., A. Fuller, and A. Munro (2004), "Workplace learning in context". Psychology Press.

[364] Ramachandran, S., S.V. Rao, and T. Goles (2008), "Information security cultures of four professions: a comparative study". in Hawaii International Conference on System Sciences, Proceedings of the 41st Annual. IEEE.

[365] Ramsden, J.M. (1992), "If It's Enjoyable, Is It Science?". School Science Review, volume 73, issue 265, pages 65-71.

[366] Ransbotham, S. (2010), "An Empirical Analysis of Exploitation Attempts Based on Vulnerabilities in Open Source Software". in Proceedings of the 9th Workshop on Economics of Information Security, Cambridge, MA, June 2010.

[367] Ransbotham, S. (2010), "An Empirical Analysis of Exploitation Attempts Based on Vulnerabilities in Open Source Software". in WEIS.

[368] Raymond, E. (1999), "The cathedral and the bazaar". Knowledge, Technology & Policy, volume 12, issue 3, pages 23-49.

[369] Razzaq, A., et al. (2014), "Ontology for attack detection: An intelligent approach to web application security", computers & security. pages 124-146.

[370] Redwine Jr, S.T. (2007), "Software assurance: A curriculum guide to the common body of knowledge to produce, acquire, and sustain secure software". Homeland Security.

[371] Reich, B.H., A. Gemino, and C. Sauer (2008), "Modeling the knowledge perspective of IT projects". PMI Research Conference, Warsaw, Poland.

[372] Ripoche, G. and L. Gasser (2003), "Scalable automatic extraction of process models for understanding F/OSS bug repair". in Proceedings of ICSSEA'03.

[373] Rivet, A.E. and J. Krajcik (2008), "Contextualizing instruction: Leveraging students' prior knowledge and experiences to foster understanding of middle school science". Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching, volume 45, issue 1, pages 79-100.

[374] Roberts, J. (2000), "From know-how to show-how? Questioning the role of information and communication technologies in knowledge transfer". Technology Analysis & Strategic Management, volume 12, issue 4, pages 429-443.

[375] Roberts, J.A., I.-H. Hann, and S.A. Slaughter (2006), "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects". Management science, volume 52, issue 7, pages 984-999.

[376] Rosa, M.G., M.R. Borges, and F.M. Santoro (2003), "A conceptual framework for analyzing the use of context in groupware", in Groupware: Design, Implementation, and Use, Springer. pages 300-313.

[377] Rosenberg, M.J. (2005), "Beyond e-learning: Approaches and technologies to enhance organizational knowledge, learning, and performance". volume: John Wiley & Sons.

[378] Ross, R., J.C. OREN, and M. McEvilley (2014), "Systems Security Engineering". NIST Special Publication.

[379] Rouse, W.B. and N.M. Morris (1986), "On looking into the black box: Prospects and limits in the search for mental models". Psychological bulletin, volume 100, issue 3, pages 349.

[380] Ruiz, R. (2019), "A Study of the UK Undergraduate Computer Science Curriculum: A Vision of Cybersecurity". in 2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3). IEEE.

[381] Rus, I. and M. Lindvall (2002), "Knowledge management in software engineering". IEEE software, volume 19, issue 3, pages 26.

[382] Ryan, R.M. and E.L.J.A.p. Deci (2000), "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being". volume 55, issue 1, pages 68.

[383] Ryoo, J., et al. (2016), "The use of security tactics in open source software projects". IEEE Transactions on Reliability, volume 65, issue 3, pages 1195-1204.

[384] Ryoo, J., A. Techatassanasoontorn, and D. Lee (2009), "Security education using second life". IEEE Security & Privacy, volume 7, issue 2, pages 71-74.

[385] Safa, N.S. and R. Von Solms (2016), "An information security knowledge sharing model in organizations". Computers in Human Behavior, volume 57, issue, pages 442-451.

[386] SAFECode, "Fundamental practices for secure software development"; Available from: https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf. (Accessed on Mar. 3, 2019)

[387] Saito, M., et al. (2015), "A case-based management system for secure software development using software security knowledge". Procedia computer science, volume 60, issue, pages 1092-1100.

[388] Salini, P. and S. Kanmani (2013), "Ontology-based representation of reusable security requirements for developing secure web applications". International Journal of Internet Technology and Secured Transactions, volume 5, issue 1, pages 63-83.

[389] Saltzer, J.H. and M.D. Schroeder (1975), "The protection of information in computer systems". Proceedings of the IEEE, volume 63, issue 9, pages 1278-1308.

[390] Scacchi, W. (2002), "Understanding the requirements for developing open source software systems". in IEE Proceedings--Software. IET.

[391] Scacchi, W., et al. (2006), "Understanding free/open source software development processes". Software Process: Improvement and Practice, volume 11, issue 2, pages 95-105.

[392] Schaeffer, R. (2010), "National information assurance (ia) glossary". CNSS Secretariat, NSA, Ft. Meade.

[393] Schein, E.H. (1990), "Organizational culture". volume 45. American Psychological Association.

[394] Schilpp, P.A. (1974), "The Philosophy of Karl Popper". volume 2. Open Court LaSalle, IL.

[395] Schlienger, T. and S. Teufel (2002), "Information security culture", in Security in the Information Society, Springer. pages 191-201.

[396] Schlienger, T. and S. Teufel (2003), "Analyzing information security culture: increased trust by an appropriate information security culture". in 14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.: IEEE.

[397] Schlienger, T. and S. Teufel (2003), "Analyzing information security culture: increased trust by an appropriate information security culture". in Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on. IEEE.

[398] Schlienger, T. and S. Teufel (2005), "Tool supported management of information security culture". in IFIP International Information Security Conference. Springer.

[399] Schmidt, D.C. and A. Porter (2001), "Leveraging open-source communities to improve the quality & performance of open-source software". in Proceedings of the 1st Workshop on Open Source Software Engineering. Citeseer.

[400] Schryen, G. and R. Kadura (2009), "Open source vs. closed source software: towards measuring security". in Proceedings of the 2009 ACM symposium on Applied Computing.

[401] Schweitzer, D. and W. Brown (2009), "Using visualization to teach security". Journal of Computing Sciences in Colleges, volume 24, issue 5, pages 143-150.

[402] Seel, N.M., S. Al-Diban, and P. Blumschein (2000), "Mental models & instructional planning", in Integrated and holistic perspectives on learning, instruction and technology, Springer. pages 129-158.

[403] SEI-CERT, "SEI CERT C secure coding standard"; Available from: https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards. (Accessed on Mar. 5, 2019)

[404] Seifert, C.M. and E.L. Hutchins (1988), "Learning from error". American Society for Engineering Education Washington DC

[405] Shambaugh, N. (1995), "The cognitive potentials of visual constructions". Journal of Visual Literacy, volume 15, issue 1, pages 7-24.

[406] Sharma, S., V. Sugumaran, and B.J.I.S.J. Rajagopalan (2002), "A framework for creating hybrid-open source software communities". volume 12, issue 1, pages 7-25.

[407] Sharp, H., Y. Dittrich, and C.R. de Souza (2016), "The role of ethnographic studies in empirical software engineering". IEEE Transactions on Software Engineering, volume 42, issue 8, pages 786-804.

[408] Shaw, R.S., et al. (2009), "The impact of information richness on information security awareness training effectiveness". volume 52, issue 1, pages 92-100.

[409] Sheeran, P. and S.J.J.o.A.S.P. Orbell (1999), "Augmenting the theory of planned behavior: roles for anticipated regret and descriptive norms 1". volume 29, issue 10, pages 2107-2142.

[410] Sholler, D., "Community Call – Governance strategies for open source research software projects"; Available from: https://www.r-bloggers.com/community-call-governance-strategies-for-open-source-research-software-projects/. (Accessed on March 3, 2019)

[411] Shrivastav, H. and S.R. Hiltz (2013), "Information overload in technology-based education: A meta-analysis".

[412] Shuaibu, B.M., et al. (2015), "Systematic review of web application security development model". volume 43, issue 2, pages 259-276.

[413] Silic, M. and A. Back (2016), "The influence of risk factors in decision-making process for open source software adoption". International Journal of Information Technology & Decision Making, volume 15, issue 01, pages 151-185.

[414] Simon, H.A. (1988), "The science of design: creating the artificial", Design Issues. pages 67-82.

[415] Singh, M.P.J.A.T.o.I.S. and Technology (2013), "Norms as a basis for governing sociotechnical systems". volume 5, issue 1, pages 21.

[416] Singh, V. and L. Holt (2013), "Learning and best practices for learning in open-source software communities". Computers & Education, volume 63, issue, pages 98-108.

[417] Siponen, M., et al. (2014), "Employees' adherence to information security policies: An exploratory field study". volume 51, issue 2, pages 217-224.

[418] Siponen, M.T.J.I.M. and C. Security (2000), "A conceptual foundation for organizational information security awareness". volume 8, issue 1, pages 31-41.

[419] Skuce, D. and I. Meyer (1990), "Concept analysis and terminology: a knowledge-based approach to documentation". in Proceedings of the 13th conference on Computational linguistics-Volume 1. Association for Computational Linguistics.

[420] Snyk, "The state of open source security - 2019"; Available from: https://snyk.io/opensourcesecurity-2019/. (Accessed on Novemver 2, 2019)

[421] Sodiya, A.S., S.A. Onashoga, and O. Ajayī (2006), "Towards Building Secure Software Systems". Issues in Informing Science & Information Technology, volume 3.

[422] Sowe, S.K., R. Ghosh, and L. Soete (2009), "Annals of Knowledge Sharing in Distributed Software Development Environments: Experience from Open Source Software Projects". in Software Engineering (Workshops).

[423] Sowe, S.K., A. Karoulis, and I. Stamelos (2006), "A constructivist view of knowledge management in open source virtual communities", in Managing learning in virtual settings: the role of context, IGI Global. pages 290-308.

[424] Sowe, S.K., I. Stamelos, and L. Angelis (2008), "Understanding knowledge sharing activities in free/open source software projects: An empirical study". Journal of Systems and Software, volume 81, issue 3, pages 431-446.

[425] Specht, M. (2008), "Designing contextualized learning", in Handbook on information technologies for education and training, Springer. pages 101-111.

[426] Stack Overflow, "Developer Survey Results 2018"; Available from: https://insights.stackoverflow.com/survey/2018. (Accessed on May 2, 2019)

[427] Stallings, W., et al. (2012), "Computer security: principles and practice". volume: Pearson Education Upper Saddle River, NJ, USA.

[428] Stol, K.-J. and M.A. Babar (2009), "Reporting empirical research in open source software: the state of practice". in IFIP International Conference on Open Source Systems. Springer.

[429] Straub, D.W. (1989), "Validating instruments in MIS research". MIS quarterly, volume, issue, pages 147-169.

[430] Sundqvist, J. (2018), "Reasons for lacking web security: An investigation into the knowledge of web developers".

[431] Swan, J.A. and T.G. Spiro (1995), "Context in chemistry: Integrating environmental chemistry with the chemistry curriculum". ACS Publications.

[432] Syed, R. and H. Zhong (2018), "Cybersecurity Vulnerability Management: An Ontology-Based Conceptual Model".

[433] Tan, L., et al. (2014), "Bug characteristics in open source software". Empirical Software Engineering, volume 19, issue 6, pages 1665-1705.

[434] Tawileh, A., J. Hilton, and S. Mcintosh (2006), "Modelling the Economics of Free and Open Source Software Security". in ISSE 2006 - Securing Electronic Business Processes: Highlights of the Information Security Solutions Europe Conference. Springer.

[435] Taylor, B. and S. Azadegan (2006), "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. ACM.

[436] Teddlie, C. and A. Tashakkori (2009), "Foundations of mixed methods research: Integrating quantitative and qualitative approaches in the social and behavioral sciences". volume: Sage.

[437] Teece, D.J., G. Pisano, and A. Shuen (1997), "Dynamic capabilities and strategic management". Strategic management journal, pages 509-533.

[438] Thomas, T.W., et al. (2018), "Security during application development: An application security expert perspective". in Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.

[439] Tilton, J., "What is an Infostructure"; Available from: http://library.creatifica.com/information-architecture/infostructure-coining-the-term.pdf. (Accessed on Jul, 2, 2018)

[440] Tiwana, A. and E. McLean (2002), "Knowledge integration and individual expertise development in e-business project teams: prom the pod to the peas". in Proceedings of the 2002 ACM SIGCPR conference on Computer personnel research. ACM.

[441] Trainer, E., et al. (2005), "Bridging the gap between technical and social dependencies with ariadne". in Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange. ACM.

[442] Trist, E.L. and K.W. Bamforth (1951), "Some social and psychological consequences of the longwall method of coal-getting: An examination of the psychological situation and defences of a work group in relation to the social structure and technological content of the work system". Human relations, volume 4, issue 1, pages 3-38.

[443] Tsoumas, B. and D. Gritzalis (2006), "Towards an ontology-based security management". in Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on. IEEE.

[444] Tudorache, T., et al. (2013), "WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web". Semantic web, volume 4, issue 1, pages 89-99.

[445] Tyre, M.J. and E. Von Hippel (1997), "The situated nature of adaptive learning in organizations". Organization science, volume 8, issue 1, pages 71-83.

[446] Uschold, M. and M. Gruninger (1996), "Ontologies: Principles, methods and applications". The knowledge engineering review, volume 11, issue 2, pages 93-136.

[447] Vadalasetty, S.R. (2003), "Security concerns in using open source software for enterprise requirements". SANS Institute.

[448] Vaishnavi, V. and W. Kuechler (2004), "Design science research in information systems". January, volume 20, issue, pages 2004.

[449] Valverde, S., et al. (2006), "Self-organization patterns in wasp and open source communities". IEEE Intelligent Systems, volume 21, issue 2, pages 36-40.

[450] Van den Hooff, B., et al. (2003), "Knowledge sharing in knowledge communities". in Communities and technologies. Springer.

[451] Van Niekerk, J. and R. Von Solms (2005), "A holistic framework for the fostering of an information security sub-culture in organizations". in Issa.

[452] Van Oers, B. (1998), "From context to contextualizing". Learning and instruction, volume 8, issue 6, pages 473-488.

[453] van Oorschot, P.C. (2019), "Software security and systematizing knowledge". IEEE Security & Privacy, volume 17, issue 3, pages 4-6.

[454] Vangaveeti, A. (2015), "An Assessment of Security Problems in Open Source Software".

[455] Velasco, J.L., et al. (2009), "Modelling reusable security requirements based on an ontology framework". volume 41, issue 2, pages 119.

[456] Venkatesh, V. and S.A. Brown (2001), "A longitudinal investigation of personal computers in homes: adoption determinants and emerging challenges". MIS quarterly.

[457] Veracode, "State of Software Security"; Available from: https://www.veracode.com/state-of-software-security-report/. (Accessed on May 3, 2019)

[458] Verdon, D. and G. McGraw (2004), "Risk analysis in software design". IEEE Security & Privacy, volume 2, issue 4, pages 79-84.

[459] Viega, J. and G. McGraw (2011), "Building Secure Software: How to Avoid Security Problems the Right Way ". Addison-Wesley Professional.

[460] Viega, J. and G. McGraw (2011), "Building Secure Software: How to Avoid Security Problems the Right Way (paperback)(Addison-Wesley Professional Computing Series)". volume: Addison-Wesley Professional.

[461] Von Bertalanffy, L. (1950), "The theory of open systems in physics and biology". Science, volume 111, issue 2872, pages 23-29.

[462] Von Krogh, G., et al. (2012), "Carrots and rainbows: Motivation and social practice in open source software development". MIS quarterly, volume 36, issue 2, pages 649-676.

[463] Von KROGh, G. and S. Spaeth (2007), "The open source software phenomenon: Characteristics that promote research". The Journal of Strategic Information Systems, volume 16, issue 3, pages 236-253.

[464] von Krogh, G., S. Spaeth, and S. Haefliger (2005), "Knowledge reuse in open source software: An exploratory study of 15 open source projects". in Proceedings of the 38th Annual Hawaii International Conference on System Sciences. IEEE.

[465] Von Krogh, G., S. Spaeth, and K.R. Lakhani (2003), "Community, joining, and specialization in open source software innovation: a case study". Research Policy, volume 32, issue 7, pages 1217-1241.

[466] Von Krogh, G. and E. Von Hippel (2003), "Special issue on open source software development", Elsevier. pages.

[467] Von Krogh, G. and E. Von Hippel (2006), "The promise of research on open source software". Management science, volume 52, issue 7, pages 975-983.

[468] Vos, R. (2014), "The use of context in science education".

[469] Vouk, M. and L. Williams (2013), "Using software reliability models for security assessment—Verification of assumptions". in 2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE.

[470] Walden, J., et al. (2009), "Security of open source web applications". in Proceedings of the 2009 3rd international Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society.

[471] Walker, G.H., et al. (2008), "A review of sociotechnical systems theory: a classic concept for new command and control paradigms". Theoretical issues in ergonomics science, volume 9, issue 6, pages 479-499.

[472] Walls, J.G., G.R. Widmeyer, and O.A. El Sawy (1992), "Building an information system design theory for vigilant EIS". Information systems research, volume 3, issue 1, pages 36-59.

[473] Wand, Y., V.C. Storey, and R. Weber (1999), "An ontological analysis of the relationship construct in conceptual modeling". ACM Transactions on Database Systems (TODS), volume 24, issue 4, pages 494-528.

[474] Wang, X., et al. (2004), "Semantic space: An infrastructure for smart spaces". IEEE Pervasive computing, volume 3, issue 3, pages 32-39.

[475] Wasko, M.M. and S. Faraj (2005), "Why should I share? Examining social capital and knowledge contribution in electronic networks of practice". MIS quarterly, volume, issue, pages 35-57.

[476] Wason, P.C. and D. Shapiro (1971), "Natural and contrived experience in a reasoning problem". The Quarterly Journal of Experimental Psychology, volume 23, issue 1, pages 63-71.

[477] Watkins, C., et al. (2002), "Effective learning", Institute of Education, University of London. pages.

[478] Webster, J. and R.T. Watson (2002), "Analyzing the past to prepare for the future: Writing a literature review". MIS quarterly, pages xiii-xxiii.

[479] Weick, K.E. (1989), "Theory construction as disciplined imagination". Academy of management review, volume 14, issue 4, pages 516-531.

[480] Weill, P. and R. Woodham (2002), "Don't just lead, govern: Implementing effective IT governance".

[481] Wen, S.-F. (2017), "Software Security in Open Source Development: A Systematic Literature Review". in Proceedings of the 21st Conference of Open Innovations Association (FRUCT). IEEE.

[482] Wen, S.-F. (2018), "An Empirical Study on Security Knowledge Sharing and Learning in Open Source Software Communities". Computers volume 7, issue 4, pages 49.

[483] Wen, S.-F. (2018), "Learning secure programming in open source software communities: a socio-technical view". in Proceedings of the 6th International Conference on Information and Education Technology. ACM.

[484] Wen, S.-F. and B. Katt (2019), "Development of Ontology-Based Software Security Learning System with Contextualized Learning Approaches". Journal of Advances in Information Technology volume 10, issue 3, pages 1-10.

[485] Wen, S.-F. and B. Katt (2019), "Learning Software Security in Context: An Evaluation in Open Source Software Development Environment", Proceedings of the 14th International Conference on Availability, Reliability and Security, ACM. pages 1-10.

[486] Wen, S.-F. and B. Katt (2019), "Managing Software Security Knowledge in Context: An Ontology Based Approach". Information, volume 10, issue 6, pages 216.

[487] Wen, S.-F. and B. Katt (2019), "Preliminary Evaluation of an Ontology-Based Contextualized Learning System for Software Security", Proceedings of the Evaluation and Assessment on Software Engineering, ACM. pages 90-99.

[488] Wen, S.-F. and B. Katt (2019), "Preliminary Evaluation of an Ontology-Based Contextualized Learning System for Software Security". in Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering, EASE 2019, April 14-17, 2019. Copenhagen, Denmark.

[489] Wen, S.-F. and B. Katt (2019), "Towards a Context-Based Approach for Software Security Learning". Journal of Applied Security Research, volume 15, issue 2, pages 1-20.

[490] Wen, S.-F., K. Mazaher, and K. Stewart (2019), "An Empirical Study of Security Culture in Open Source Software Communities". in 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE.

[491] Wenger, E. (2000), "Communities of practice and social learning systems". Organization, volume 7, issue 2, pages 225-246.

[492] Whitney, M., et al. (2015), "Embedding secure coding instruction into the IDE: A field study in an advanced CS course". in Proceedings of the 46th ACM Technical Symposium on Computer Science Education. ACM.

[493] Whitworth, B. (2009), "Socio-technical Design and Social Networking Systems, chapter The Social Requirements of Technical Systems", IGI Global.

[494] Wikia, "Psychology Wiki"; Available from: http://psychology.wikia.com/wiki/Knowledge_structure. (Accessed on Nov. 13, 2018)

[495] Williams, J. and D. Wichers (2017), "The ten most critical web application security risks". OWASP Foundation.

[496] Williams, K.A., et al. (2014), "Teaching secure coding for beginning programmers". Journal of Computing Sciences in Colleges, volume 29, issue 5, pages 91-99.

[497] Winter, R. (2008), "Design science research in Europe". European Journal of Information Systems, volume 17, issue 5, pages 470-475.

[498] Witschey, J. (2013), "Secure development tool adoption in open-source". in Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity. ACM.

[499] Woon, I.M. and A.J.I.J.o.H.-C.S. Kankanhalli (2007), "Investigation of IS professionals' intention to practise secure development of applications". volume 65, issue 1, pages 29-41.

[500] Workman, M., D.C. Phelps, and J.N. Gathegi (2012), "Information security for managers". Jones & Bartlett Publishers.

[501] Wu, L.-W. and J.-R. Lin (2013), "Knowledge sharing and knowledge effectiveness: learning orientation and co-production in the contingency model of tacit knowledge". Journal of Business & Industrial Marketing, volume 28, issue 8, pages 672-686.

[502] Xiao, S., J. Witschey, and E. Murphy-Hill (2014), "Social influences on secure development tool adoption: why security tools spread". in Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing. ACM.

[503] Xie, J., H.R. Lipford, and B. Chu (2011), "Why do programmers make security errors?". in Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on. IEEE.

[504] Xiong, M., et al. (2004), "Perspectives on the Security of Open Source Software".

[505] Yamauchi, Y., et al. (2000), "Collaboration with Lean Media: how open-source software succeeds". in Proceedings of the 2000 ACM conference on Computer supported cooperative work. ACM.

[506] Ye, Y. and K. Kishida (2003), "Toward an understanding of the motivation Open Source Software developers". in Proceedings of the 25th international conference on software engineering. IEEE Computer Society.

[507] Ye, Y., Y. Yamamoto, and K. Nakakoji (2007), "A socio-technical framework for supporting programmers". in Proceedings of the the 6th joint meeting of the European

software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM.

[508] Young, A., "Too Much Information: Ineffective Intelligence Collection", Harvard Internaton Review; Available from: https://hir.harvard.edu/too-much-information/. (Accessed on Oct. 13, 2019)

[509] Yuan, X., et al. (2010), "Visualization tools for teaching computer security". ACM Transactions on Computing Education (TOCE), volume 9, issue 4, pages 20.

[510] Yuan, X., et al. (2016), "Secure software engineering education: Knowledge area, curriculum and resources". Journal of Cybersecurity Education, Research and Practice, volume 2016, issue 1, pages 3.

[511] Yue, C. (2016), "Teaching computer science with cybersecurity education built-in". in 2016 {USENIX} Workshop on Advances in Security Education ({ASE} 16).

[512] Zack, M.H. (1999), "Managing codified knowledge". Sloan management review, volume 40, issue 4, pages 45-58.

[513] Zadeh, J. and D. DeVolder (2007), "Software development and related security issues". in Proceedings 2007 IEEE SoutheastCon. IEEE.

[514] Zeitlyn, D.J.R.p. (2003), "Gift economies in the development of open source software: anthropological reflections". volume 32, issue 7, pages 1287-1291.

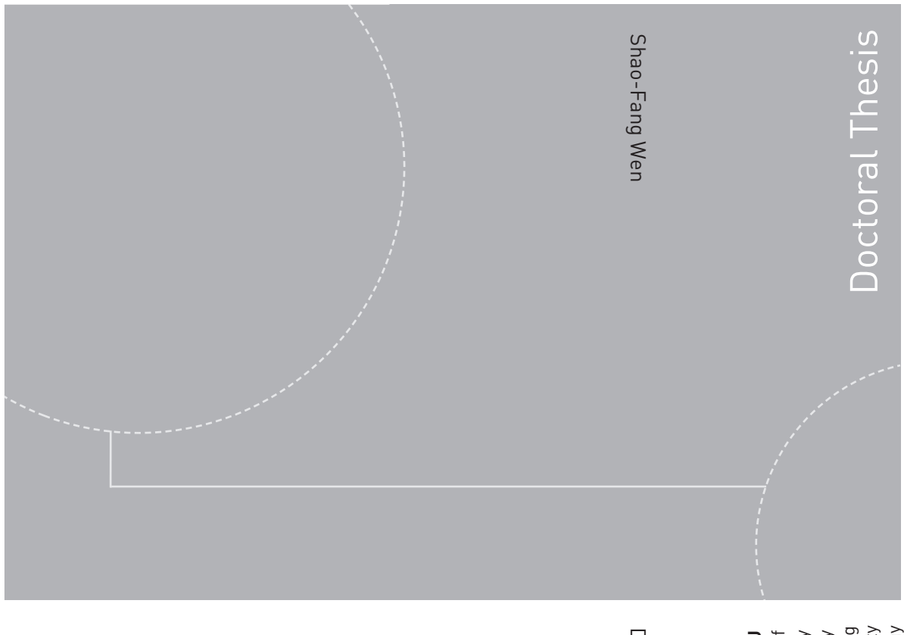
[515] Zelkowitz, M.V. and D.R. Wallace (1998), "Experimental models for validating technology". Computer, volume 31, issue 5, pages 23-31.

[516] Zevin, S. (2009), "Standards for security categorization of federal information and information systems". volume: DIANE Publishing.

[517] Zhu, J., H.R. Lipford, and B. Chu (2013), "Interactive support for secure programming education". in Proceeding of the 44th ACM technical symposium on Computer science education. ACM.

[518] Zikmund, W.G., et al. (2013), "Business research methods". Cengage Learning.

ISBN 978-82-326-4650-0 (printed version)
ISBN 978-82-326-4651-7 (electronic version)
ISSN 1503-8181



Doctoral theses at NTNU, 2020:151

Shao-Fang Wen

A Multi-Discipline Approach for Enhancing Developer Learning in Software Security

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology
and Electrical Engineering
Department of Information Security
and Communication Technology

NTNU
Norwegian University of
Science and Technology

Doctoral theses at NTNU, 2020:151

NTNU

NTNU
Norwegian University of
Science and Technology