

Md Musfiqur Rahman Milton

DESIGN AND ASSESSMENT OF DEVSECOPS EXERCISE FOR SECURE SOFTWARE DEVELOPMENT EDUCATION

Master's Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Tiina Schafeitel-Tähtinen
Marko Helenius
May 2025

ABSTRACT

Md Musfiqur Rahman Milton: Design and Assessment of DevSecOps exercise for Secure Software Development Education

Master's Thesis

Tampere University

Master of Science (Technology), Information Technology

May 2025

Modern software systems increasingly depend on secure development practices. However, many academic curricula do not adequately address the industry-wide shift toward DevSecOps, which integrates security throughout the software development lifecycle. This thesis aimed to bridge that gap by designing and empirically evaluating a hands-on exercise that introduces students to DevSecOps within a realistic development environment.

The exercise involved students constructing a development pipeline using tools such as Jenkins, SonarQube, Snyk, Trivy, and OWASP ZAP. The OWASP Juice Shop, an intentionally vulnerable web application, served as the testbed for this exercise. Within this framework, students engaged in various levels of security testing, including static analysis, dynamic analysis, software composition analysis, and container scanning. Additionally, they explored key principles of DevSecOps, including early integration, automation, and shared responsibility.

Evaluation was conducted through pre- and post-exercise surveys completed by 75 students enrolled in a university-level secure programming course. The quantitative analysis revealed statistically significant improvements in both knowledge and self-efficacy, while qualitative analysis indicated increased confidence and practical insight. Although students began with high attitudes and interest in secure development, the exercise successfully maintained that consistency and produced outcomes that closely aligned with their initial expectations. Notably, these results were consistent among participants, irrespective of age, gender, educational background, or prior experience.

This thesis presents a replicable and scalable model for incorporating DevSecOps into software engineering education. By merging real-world technical skills with an effective teaching approach, it enhances our understanding of how to teach secure development practices. Additionally, it provides valuable insights for educators, curriculum planners, and universities seeking to better align their programs with industry demands.

Keywords: DevSecOps, SDLC, Software Security, DevSecOps Education, SAST, CI/CD

The originality of this thesis has been verified using the Turnitin Originality Check service.

USE OF AI IN THIS THESIS

This document I have utilised Artificial Intelligence (AI) tools in my thesis:

- No
- Yes

I hereby declare that following AI-based tools were used in this thesis:

Application	Version
OpenAI ChatGPT	4o
QuillBot	Premium

Purpose of using AI tools:

- Assisting with grammar and readability.
- Maintaining fluency and coherence in the content.

Parts where AI was used:

- Assisting with grammar and readability in Chapter 1 and 2

I, the author of this thesis, unequivocally acknowledge and bear responsibility for all content presented, including areas where artificial intelligence was used. Although AI assisted with paragraph coherence, grammatical corrections, and formatting, all content-related decisions, including the selection of scientific publications, the inclusion of concepts, and the thesis structure, were made independently. I accept full accountability for any violations of ethical or academic standards resulting from the usage of AI.

PREFACE

First and foremost, I express my deepest gratitude to my Creator, Allah (SWT), whose guidance and mercy have carried me through every step of this journey. Without His will, this work would not have been possible.

I am deeply grateful to my supervisors, Tiina Schafeitel-Tähtinen and Dr. Marko Helenius, for their guidance, patience, and thoughtful feedback throughout the thesis process. Their support, both in reviewing my drafts and in shaping the direction of this research, has been invaluable. I especially thank Dr. Helenius for allowing me to develop and implement the DevSecOps exercise in his course, which became a central part of this study.

I am also truly thankful to my parents, whose prayers, support, unwavering belief in me and quiet strength have always been my foundation. Their support has been behind every accomplishment in my life.

To my fiancée, Fariha, thank you for being my calm in the chaos. Your constant encouragement and emotional support have meant more to me than words can express.

This work stands as a reflection of the support, commitment, and kindness I've received throughout this journey. I am grateful to everyone who contributed to its completion.

Md Musfiqur Rahman Milton

Tampere, Finland

31st May 2025

CONTENTS

1. INTRODUCTION	1
1.1 Motivation	1
1.2 Scopes and Goals of the Thesis	2
1.3 Research Questions	2
1.4 Contributions to Scientific Research	3
1.5 Structure of the Thesis	4
2. DEVSECOPS AND SECURE DEVELOPMENT	5
2.1 Software Development Life Cycle (SDLC)	5
2.1.1 Phases of SDLC	5
2.1.2 Models of SDLC	7
2.2 DevOps and SDLC	9
2.3 Overview of DevSecOps	10
2.3.1 Principles of DevSecOps	11
2.3.2 Key Practices and Tools in DevSecOps	12
2.3.3 Stages of DevSecOps Lifecycle	14
2.4 Common Vulnerabilities: OWASP Top 10	15
2.5 Importance of DevSecOps in Education	16
2.6 Theoretical Background of Assessment Variables	17
2.7 Linking Assessment Variables to Survey Design	20
3. RELATED WORK	22
3.1 Search Process	22
3.2 DevSecOps and Security	23
3.3 Approaches in Software Security Education	24
3.4 DevSecOps in Secure Software Development Education	26
4. EXERCISE DESIGN AND IMPLEMENTATION	27
4.1 Methodology	27
4.2 Analysis of Exercise Outputs	28
4.2.1 Overview of CI/CD Pipeline	28
4.2.2 Static Analysis	29
4.2.3 Software Composition Analysis	31
4.2.4 File System and Image Scanning	31
4.2.5 Dynamic Testing	32
5. ASSESSMENT DESIGN AND ANALYSIS	34
5.1 Methodology	34
5.1.1 Participant Recruitment	34

5.1.2	Survey Design and Structure	34
5.1.3	Data Collection and Preprocessing	36
5.1.4	Data Analysis	36
5.2	Quantitative Analysis	38
5.2.1	Reliability Testing	38
5.2.2	Normality Testing	39
5.2.3	Hypothesis Testing and Effect Size	39
5.2.4	Hypothesis Testing on Demographic Information	50
5.3	Qualitative Analysis	55
5.3.1	Helpful Aspects of the Exercise	55
5.3.2	Scope of Improvement	57
5.3.3	Key Takeaways from the Exercise	58
6.	DISCUSSION.....	60
6.1	Addressing the Research Questions.....	60
6.2	Comparison to Related Work.....	61
7.	CONCLUSIONS.....	65
7.1	Limitations and Future Work	66
	REFERENCES.....	68
	APPENDIX A: SURVEY QUESTIONS	77

LIST OF FIGURES

<i>Figure 2.1. DevOps Lifecycle and Continuous Delivery Process [28]</i>	9
<i>Figure 2.2. DevSecOps Lifecycle and Continuous Delivery Process [31]</i>	10
<i>Figure 2.3. Stages of DevSecOps Lifecycle with Security Checkpoints [46]</i>	15
<i>Figure 4.1. Overview of the pipeline architecture</i>	27
<i>Figure 4.2. Pipeline Dashboard</i>	29
<i>Figure 4.3. Pipeline Overview with all stages</i>	29
<i>Figure 4.4. Dashboard of SonarQube Scan</i>	30
<i>Figure 4.5. Detailed view of Security Hotspots</i>	30
<i>Figure 4.6. Scan Report of Synk</i>	31
<i>Figure 4.7. Scan Report from Image Scanning</i>	32
<i>Figure 4.8. Scan Report from Dynamic Testing</i>	33
<i>Figure 5.1. Overview of the Assessment Design</i>	37
<i>Figure 5.3. Differences between pre- and post-survey responses of sKN variable</i>	41
<i>Figure 5.4. Data distribution of Interest, sIN variable</i>	42
<i>Figure 5.5. Differences between pre- and post-survey responses of sIN variable</i>	43
<i>Figure 5.6. Data distribution of Self-efficacy, sSE variable</i>	44
<i>Figure 5.7. Differences between pre- and post-survey responses of sSE variable</i>	45
<i>Figure 5.8. Data distribution of Attitude, sAT variable</i>	46
<i>Figure 5.9. Differences between pre- and post-survey responses of sAT variable</i>	47
<i>Figure 5.10. Data distribution of Expectations/Outcomes, sEM variable</i>	48
<i>Figure 5.11. Differences between pre- and post-survey responses of sEM variable</i>	49
<i>Figure 5.12. Frequency of themes for the Helpful Aspects of the Exercise</i>	56
<i>Figure 5.13. Frequency of themes for the Scopes of Improvement</i>	57
<i>Figure 5.14. Frequency of themes for the Key Takeaways from the Exercise</i>	58

LIST OF TABLES

<i>Table 3.1. Search strings for relevant work.....</i>	22
<i>Table 5.1. Results of the Reliability Test.....</i>	39
<i>Table 5.2. Results of the Normality Test.....</i>	39
<i>Table 5.3. Results of the Hypothesis Testing and Effect Size Calculations.....</i>	49
<i>Table 5.4. Hypothesis Testing on Demographic Information for Knowledge.....</i>	50
<i>Table 5.5. Hypothesis Testing on Demographic Information for Interest.....</i>	51
<i>Table 5.6. Hypothesis Testing on Demographic Information for Self-efficacy.....</i>	52
<i>Table 5.7. Hypothesis Testing on Demographic Information for Attitude.....</i>	53
<i>Table 5.8. Hypothesis Testing on Demographic Information for Expectations and Outcomes.....</i>	54
<i>Table 5.9. Generated codes and themes for the Helpful Aspects of the Exercise.....</i>	56
<i>Table 5.10. Generated codes and themes for the Scopes of Improvement.....</i>	57
<i>Table 5.11. Generated codes and themes for the Key Takeaways from the Exercise.....</i>	58

LIST OF SYMBOLS AND ABBREVIATIONS

CD	Continuous Deployment
CI	Continuous Integration
CI/CD	Continuous Integration/Continuous Deployment
DAST	Dynamic Application Security Testing
DevOps	Development and Operations
DevSecOps	Development, Security, and Operations
IaC	Infrastructure as Code
ICT	Information and Communication Technology
JWT	JSON Web Token
OWASP	Open Worldwide Application Security Project
RASP	Runtime Application Self-Protection
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SBOM	Software Bill of Materials
SDD	Software Design Document
SDLC	Software Development Life Cycle
SRS	Software Requirements Specification
URL	Uniform Resource Locator
VM	Virtual Machine

N	Sample Size
p	Probability
Z	Z-score
d	Effect Size
sKN_pre	Knowledge in pre-survey
sKN_post	Knowledge in post-survey
sIN_pre	Interest in pre-survey
sIN_post	Interest in post-survey
sSE_pre	Self-efficacy in pre-survey
sSE_post	Self-efficacy in post-survey
sAT_pre	Attitude in pre-survey
sAT_post	Attitude in post-survey
sEM_pre	Expectations in pre-survey
sEM_post	Outcomes in post-survey

1. INTRODUCTION

This chapter provides an overview of the thesis by introducing its background, motivation, and objectives. It begins by explaining the motivation for the research, highlighting the growing importance of security in modern software development and the corresponding educational challenges. The chapter then defines the theoretical and practical scope of the work, outlines the key research questions, summarizes the main contributions, and presents the overall structure of the thesis.

1.1 Motivation

Software systems are now essential in almost every aspect of daily life, including healthcare, finance, transportation, and communications [1]. Because they are interconnected and sophisticated, they pose an elevated attack surface for malicious entities. With the increasing uses of software, the necessity for robust security measures has significantly increased. Attacks on cyberspace are becoming more frequent and sophisticated [2]. They have caused significant reputational and financial damage to organizations around the world. The SolarWinds and MOVEit attacks, as well as the Equifax breach, highlight the inherent vulnerability of software systems and their consequences [3-5].

The conventional approach to software development considers security an afterthought late in the development process. This approach allows vulnerabilities to persist. As a result, it may lead to their exploitation in the production environment [6]. Such a strategy is not sufficient to address the complex nature of modern cyberattacks. For overcoming this problem, the DevSecOps paradigm has been suggested. It integrates security practices into the software development lifecycle. DevSecOps highlights the "shift-left" practice, wherein security controls are added during the early phase of the development life cycle. Identifying and fixing security issues during the development life cycle lowers the chances of the vulnerabilities entering production environments. DevSecOps also promotes a culture of shared responsibility within the development, security, and operations teams. Effective adoption of DevSecOps calls for a workforce that is skilled in development as well as security practices. [7, 8]

Despite growing industry adoption of DevSecOps, a significant gap persists in software engineering education [9, 10]. Many academic programs still emphasize traditional de-

velopment practices with minimal exposure to integrated security practices [11]. Consequently, most graduates lack the necessary skills to address security integration issues in the industry. Additionally, the rapid rate of evolution in development tools and practices outpaces the changes in academic programs and hence increases the skills gap. Industry reports also indicate an apparent shortage of individuals with required skills in DevSecOps. [12]

This thesis seeks to address this education gap by designing, implementing, and evaluating a hands-on exercise on secure software development. The exercise was developed based on the principles of DevSecOps.

1.2 Scopes and Goals of the Thesis

The scope of this thesis spans both theoretical and practical dimensions of secure software development education. On the theoretical side, the study examines the phases of the SDLC and explores how security can be integrated throughout the process. It also discusses the key principles of DevSecOps. Then this work outlines common vulnerabilities as identified by the OWASP Top 10. In addition, it identifies the key variables used to evaluate educational impact, such as participants' knowledge, interest, self-efficacy, and attitudes. The practical scope and goals of this thesis are:

- To develop an exercise that integrates principles and tools of DevSecOps.
- To assess the effectiveness of the exercise in the context of secure software development education.

By aligning these theoretical and practical goals, the thesis aims to provide a replicable educational framework that supports both conceptual understanding and practical skill development in DevSecOps. The ultimate goal is to contribute to a more robust and security-conscious approach to software engineering education.

1.3 Research Questions

To structure the investigation and ensure alignment with the thesis objectives, a set of research questions has been formulated. These questions are intended to examine both the design of the exercise and its broader implications. Drawing from current challenges in secure software development education, the questions aim to explore how DevSecOps principles can be effectively integrated into a learning environment, and how such integration influences key learner outcomes. Together, they provide a foundation

for systematically assessing the relevance, impact, and inclusivity of the proposed approach. The research questions are:

- RQ1:** How can an exercise be designed to incorporate DevSecOps concepts into software development education?
- RQ2:** How does the DevSecOps exercise affect participants' knowledge, interest, self-efficacy, and attitudes toward secure software development?
- RQ3:** To what extent do participants' initial expectations correspond to their perceived outcomes following the exercise?
- RQ4:** How do demographic variables—such as age, gender, educational background, and prior experience—affect the impact of the exercise?

RQ1 is to determine the systematic design and development of the exercise. The solution to this question is found by determining the fundamental principles of DevSecOps and the applicable security practices and tools. It is also necessary to develop a structured exercise manual that can be incorporated into existing software development curricula. RQ2, the second question, identifies the effect of the exercise on participants. In particular, it assesses measurable changes in the participants' knowledge and interest in secure development practices. It explores their self-efficacy for implementing these practices and general attitudes toward security integration within software development. Answering this question serves as empirical evidence in support of the value and effectiveness of the exercise.

RQ3 investigates the correlation between participants' expectations and their experience upon finishing the exercise. By contrasting expectations with perceived learning outcomes, this question assesses whether the exercise is fulfilling participants' needs and expectations. RQ4 investigates to what degree demographic variables influence participants' knowledge, interest, self-efficacy, attitude, and perceived outcomes. It seeks to determine patterns in how various groups benefit from the exercise.

1.4 Contributions to Scientific Research

This thesis contributes to scientific research by addressing the research questions outlined in the preceding section. A brief answer to each research question is provided below, with further explanations to be discussed in Chapter 6.

1. The developed exercise included various tools and practice of DevSecOps, such as SAST, SCA, and DAST. Chapter 4 provides a further explanation of how it integrated DevSecOps principles and tools.

2. The exercise had a statistically significant impact on participants' knowledge and self-efficacy. Both interest and attitude toward security integration were at a higher level prior to the exercise, and they remained consistent afterward.
3. Participants' expectations correspond with their perceived outcomes, since their responses remain consistent.
4. Overall, the demographic factors did not significantly affect the exercise's impact. This indicates that the exercise was accessible to all participants regardless of factors such as age, gender, education, or experience.

These contributions reflect both theoretical and empirical advances. The study demonstrates that integrating DevSecOps tools and concepts into structured educational exercises can lead to meaningful learning outcomes. Moreover, the results support the feasibility of applying such interventions in diverse academic settings, thereby informing future curriculum development and educational practice in the field of secure software engineering.

1.5 Structure of the Thesis

The rest of the thesis is structured as follows: Chapter 2 discusses the necessary theoretical information, including the Software Development Life Cycle (SDLC), principles and tools of DevSecOps, and variables for assessing the exercise. Chapter 3 presents related works in this field and identifies the research gap. Chapter 4 outlines the development of the exercise along with the results. Chapter 5 discusses the methodology for assessment design and the results of the data analysis. Chapter 6 interprets the evaluation results, addresses the research questions, and provides a comparison to related works. Finally, Chapter 7 concludes the thesis by summarizing the key findings, discussing the limitations encountered, and suggesting future work.

2. DEVSECOPS AND SECURE DEVELOPMENT

This chapter provides the theoretical foundation for this thesis. It begins by explaining the SDLC and its various phases, as well as the different SDLC models. Next, it addresses the evolution of SDLC with the introduction of DevOps, noting that security issues were not adequately addressed during this transition. The chapter then introduces DevSecOps as a solution to these issues, detailing its principles, practices, and phases. Finally, it emphasizes the significance of DevSecOps in education and explores the theoretical context of assessment variables.

2.1 Software Development Life Cycle (SDLC)

Langer, Leach, Pressman and Sommerville [13-16] describe the software development life cycle as a systematic process that development teams follow to develop cost-effective, high-quality, and secure software. It decomposes software development into various repeatable phases and offers a framework that enables companies to develop software that satisfies the requirements of the stakeholders and aligns with the expectations of the customers throughout the life cycle. Every stage of the SDLC has clearly defined objectives and deliverables that allow the process to move to the next stage of software development. Effective software development involves a successful balance of several variables, which are challenging for development teams to attain. Teams need to analyze and synthesize the requirements of multiple stakeholders, adapt to new requirements, assess the availability of resources, ensure product integration into the larger IT infrastructure, and assess how changes will impact incorporation, etc. Authors also state that SDLC comprises a collection of best practices that development teams utilize to address these considerations and create successful software. [13-16]

2.1.1 Phases of SDLC

According to Leach, Pressman and Sommerville [14-16], the software development lifecycle is typically divided into six phases, which are explained below. However, the nature of the SDLC approach can vary among different project teams. Harness.io [17] states that some teams separate planning and analysis as distinct phases in a seven-step development life cycle.

1. **Planning and Analysis:** The planning phase of a software project involves brainstorming high-level ideas about the problem, use cases, and the interface with

other programs involved in the project. The team conducts market research, feasibility testing, prototype testing, resource allocation, and collect requirements. They collect input from stakeholders, including customers, business managers and business analysts, to define the project's goals. Additionally, the team eliminates unnecessary elements to prevent the project from becoming overly complex. Requirements analysis can be segmented into different phases. A software requirements specification (SRS) document is drafted to make the team uniform during the project, detailing what the software must do, its resources, risk considerations, and time. Once the planning phase is completed, the project team knows its business objectives and related risks.

2. **Design:** Developers begin with the architectural design of the project, such as database design, user interface and navigation. There are microservices architecture in most companies, i.e., loosely connected and independently deployable pieces or services. Developers define how software will be embedded into existing company applications and services. Development teams work together to build an end user-friendly prototype and compatible with existing systems. They also produce multiple prototypes to present the product potential to stakeholders and collect feedback. A software design document (SDD) compiles this work and serves as a code development guide.
3. **Development:** This phase includes the team writing and developing software according to specification documents such as the Software Design Document (SDD) and Software Requirements Specification (SRS). The documents are used by developers to choose a suitable programming language as well as break down the project into pieces that can be managed. Other interfaces or systems, such as web pages or APIs, are created at this stage too. It is important to conduct code testing and code reviews at periodic intervals to identify bugs and vulnerabilities at this phase.
4. **Testing:** Once the development team has a working piece of software, they proceed to remove bugs and refine the end product. The quality assurance teams conduct acceptance testing, system testing, unit testing and integration testing to confirm that everything works as it should, complies with user and business needs, and works well in the company's IT infrastructure. Additionally, they inspect the software for security risks, identify how and when these risks occur, and document their results. Developers resolve bugs, apply patches, and return the program for retesting. Both automated and manual testing techniques are used, and in many cases, AI tools assist with test case generation and the analysis of test failure patterns.

5. **Deployment:** Once software has been tested and debugged, it is implemented in the production environment for user access. The goal is not only to launch the software but also to guide users on how to use it with minimal disruption to their operations. This may include activities like as beta releases, manual development, training, or on-site technical support. The goal is to ensure lower disruption to the user experience.
6. **Maintenance:** SDLC extends past software deployment, with regular maintenance and support that will need to be done to ensure its longevity. Despite rigorous testing, there are surprise fixes, new scenarios, and optimizations that will have to be resolved after deployment. Software developers publish updates, test fix patches, and resolve new bugs during this stage of the SDLC, such as rebuilding a house over time.

In summary, SDLC provides a structured framework comprising distinct yet interrelated phases that collectively support the development of robust software. While specific interpretations of the model may differ across organizations and projects, the objective remains consistent, which is to ensure that software solutions are developed in alignment with defined requirements, standards, and user expectations. [14-16]

2.1.2 Models of SDLC

The choice of an effective software development model is based on various parameters, such as the extent of the project's requirements, its complexity, and the ability of the development team. These parameters enable stakeholders to make the right decisions.

1. **Waterfall Model:** Saravanos et al. [18] and Shylesh [20] explain the waterfall model as a model in which there is linear progression through phases of SDLC. There is no overlap between phases in this model. A phase needs to be finished before the next one can be started. The model is used with projects whose requirements are clearly established and where change is minimal in development. The model offers a structure for easy documentation and tracking milestones. According to authors, the constraints of the model are that it is inflexible and does not support changes once a phase has been finished.
2. **Agile Model:** According to Martin [19] and Shylesh [20], agile is a methodology that focuses on flexibility, collaboration, and customer satisfaction. It follows the Agile Manifesto and iterates sprints of one to four weeks. Cross-functional teams perform planning, design, coding, testing, and review in these sprints. Each sprint

concludes with the delivery of a potentially shippable product with ongoing feedback and adaptation. This spiral approach allows teams to respond to changing needs and provide working software in shorter intervals than conventional methodologies. The beauty of Agile lies in the fact that it is very sensitive to change, and the defects get detected early on. However, authors also explain that this model introduces uncertainty in timelines and costs without a fixed scope and pre-planning.

3. **V-Model:** Shylesh [20] describes the V-Model, or Verification and Validation model, as a model that focuses on the connection between development tasks and testing processes. The model illustrates design and specification along one dimension and verification and validation along the other dimension. It focuses on planning test process in an early phase of the development life cycle and hence results early fault detection and removal. This is a strategy which can generate high quality software at lower cost. However, according to the author, projects with changing requirements or improve through iterative development may not suit this method.
4. **Spiral Model:** Shylesh [20] also explains the spiral model, which is an iterative risk-based software development process with four phases: planning, risk analysis, engineering, and evaluation. Spiral structure facilitates continuous product improvement. It is therefore most appropriate for large, complex projects with changing requirements and uncertainties. It promotes risk management during the development life cycle to detect defects and correct them in time. Involving the stakeholders and feedback at all levels will ensure that the final product is the intended outcome of the users. Expertise and complexity in the risk assessment process can result in higher costs. The author concludes that the added cost may be unacceptable to extremely low-scope projects.

Overall, SDLC models like Waterfall, Agile, V-Model, and Spiral provide various approaches designed for different project contexts, influenced by factors such as complexity, flexibility, and risk. These models have evolved over time to meet the changing demands of software development practices. While each model has unique strengths and limitations, their selection depends on stakeholder priorities and project characteristics. As outlined by Saravanos et al. [18], Martin [19], and Shylesh [20], selecting the appropriate model is important for aligning development efforts with project objectives and achieving efficient software delivery.

2.2 DevOps and SDLC

According to Cui [21], adopting DevOps in the SDLC represents a major milestone in software engineering. The purpose of DevOps is to minimize the gap between development and operation teams and make the software delivery process more automated and efficient.

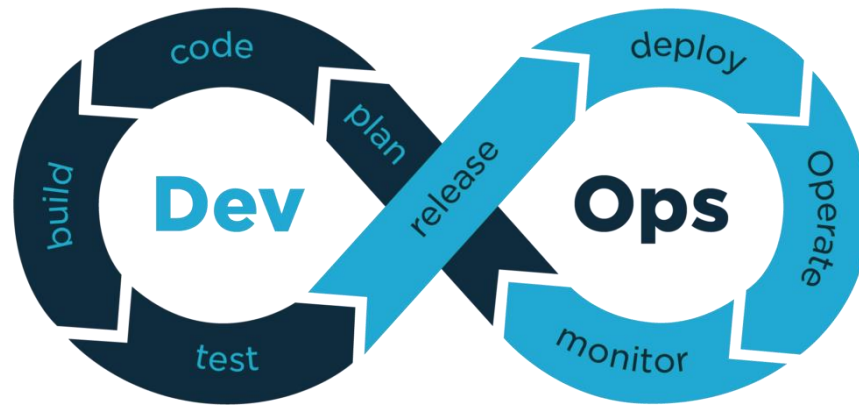


Figure 2.1. DevOps Lifecycle and Continuous Delivery Process [28]

Figure 2.1 illustrates the lifecycle of DevOps containing different phases like code and plan. Saravanos and Curinga [18] state in their research that many of the SDLC models, like waterfall and agile, indicate discrete phases of software development, which separate development, testing, and operations as silos. Zhu et al. [22] assert in their research that DevOps seeks to take down these silos through a cultural change to the continuous integration model of shared responsibility throughout the lifecycle. DevOps can even provide organizations with rapid development cycles, better code quality, and fewer unstable releases. According to Shahin et al. [23], Meyer [24], and Marijan et al. [25], some of the features that minimize the feedback cycles and eliminate the need for human involvement are automated testing, continuous delivery (CD), continuous integration (CI), and infrastructure as code (IaC). These reduce development time-to-market as well as software defects. In addition, DevOps allows for collaboration and responsibility across development and operations teams to produce scalable, reliable software. The hybrid approach can help efficiently improve the SDLC cycle. According to authors, DevOps allows for alignment between business goals and improves development cycle time and response to new technologies and market conditions [23-25].

Continuous integration (CI) is a core practice in software development with DevOps. Meyer [24] explains CI as the practice of frequently committing code changes to a com-

mon code repository, automated building and testing, and discovering and fixing integration issues in early phases. The practice allows code merging, preserves software quality, and minimizes manual complexity. It also provides real-time feedback to help teams decide how to mitigate issues. According to the author, version control tools like Git complement standard CI tools like Jenkins, Travis CI, and CircleCI and integrate well with automation across the development lifecycle.

Conversely, Continuous Deployment (CD) automatically deploy changes that have passed the testing stage into production. According to Alanda et al. [26] and Laster [27], CD allows new features, bug fixes, and improvements to be sent to consumers in nearly no time and very efficiently. It effectively shortens the cycle of building, testing, and deployment. It enables the software development teams to develop software quickly and more efficiently. Authors also found that CD increases productivity and software updates by reducing the time between code writing and deployment [26-27].

According to Rajapakse et al. [29] and Gupta [30], integrating DevOps into the SDLC brings more collaboration, automation, and speed but also has security implications. Shorter delivery cycles may sometimes push security risks to the background, which will result in applications vulnerable to attacks. Therefore, the absence of security processes clearly indicates the necessity for DevSecOps.

2.3 Overview of DevSecOps

DevSecOps is a methodology of software development that integrates security directly into the development and operations processes. According to Gupta [30], most development methodologies do not perform security checks until they have built a portion of the software.

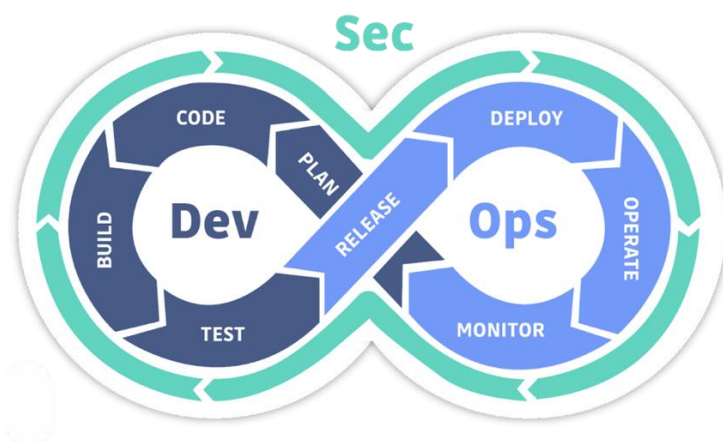


Figure 2.2. DevSecOps Lifecycle and Continuous Delivery Process [31]

According to Gupta [30], most development methodologies do not perform security checks until they have built a portion of the software. He also added that these practices resulted in delays and, simultaneously, left many vulnerabilities in the system that organizations would not identify until it was too late. Figure 2.2 illustrates the lifecycle of DevOps containing different phases like code and plan. DevSecOps fundamentally transforms the process by integrating security into the project from its start. According to Abed et al. [32] and Czekster [33], developers, operations teams, and security professionals collaborate closely enough to detect security issues early in the process. The benefit of this method is that the proactive approach enables security teams to mitigate and address any security issues as soon as they arise. This methodology uses automated security tools to consistently scan codes, run tests on applications, and assess the risk of third-party components. Authors also state that, DevSecOps enables teams to deliver software much quicker and ultimately reduce organizational risk related to cybersecurity. [32-33]

2.3.1 Principles of DevSecOps

There exists a set of principles that are important to the successful execution of DevSecOps. The principles focus on merging security into the software development lifecycle as early as possible without compromising speed or agility. According to Seghal [34] and Turner [35], these principles include several manifestos: unifying the CI/CD pipeline, enabling fail-fast automation, empowering teams to make security decisions, promoting cross-functional education, documenting thoroughly, defining relevant security checkpoints, and securing the development environments and toolchains. They are explained below.

First, unifying the CI/CD pipeline is all about embedding security initiatives, such as scans, compliance checks, and audit logs, into automated development processes. This allows security to effectively scale with the development and deployment stack. Next, fail-fast automation is designed to investigate built-in flaws as early as possible through automated testing to minimize the risk of security surprises in later stages of development. This aligns with the agile and DevOps goal of rapid iteration and feedback.

Third, empowering teams to make decisions decentralizes security responsibility and shifts accountability from solely specialized security personnel to development and operations teams, enabling them to respond faster with contextual information. Next, cross-skilling and education make sure that all team members understand the most basic se-

curity hygiene and tooling so we can create a culture of shared responsibility and awareness. Then, the importance of proper documentation is established firmly. Documentation provides an accurate trail of decisions, configurations, and threat assessments, fostering transparency and regulatory accountability.

Sixth, support for relevant checkpoints within the development pipeline is expressed. Relevant checkpoints ensure that certain security evaluations are performed when necessary while not unnecessarily blocking progress. Finally, creating and managing a secure development environment and toolchains allows for security best practices to be applied to the platforms and infrastructure throughout the development process, mitigating the risk of a compromise by using the toolchain itself.

These seven principles [34, 35] emphasize embedding security practices directly into development workflows without hindering speed or flexibility. From integrating automated checks in the CI/CD pipeline to empowering cross-functional teams, the focus lies on decentralizing security ownership, fostering a shared security mindset, and ensuring traceability through documentation. By establishing checkpoints and securing toolchains and environments, these principles support a proactive, scalable, and continuous approach to software security that aligns with the core philosophy of DevSecOps.

2.3.2 Key Practices and Tools in DevSecOps

Implementing DevSecOps effectively means utilizing many security practices and adopting them into the software development lifecycle. According to Seghal [36] and Prates et al. [37], these practices aim to identify vulnerabilities in applications at various stages of development and deployment, ensuring their security and compliance.

1. **Static Application Security Testing (SAST):** SAST is a white box testing technique that examines source code, bytecode, or binary code without executing the application. According to Seghal [38], the primary goal of SAST is to discover vulnerabilities such as SQL injections, cross-site scripting (XSS), and insecure coding practices as early in the development cycle as possible. Because it does not require a running application, SAST can be incorporated directly into the code or build phases of the CI/CD pipeline, creating an early feedback loop for developers to identify and remediate security vulnerabilities during the coding stages of development. A few popular SAST tools include SonarQube, Semgrep, Checkmarx, and Veracode SAST.
2. **Dynamic Application Security Testing (DAST):** DAST is a black box testing technique that tests the application while it is running. According to Seghal [38],

unlike SAST, DAST can be performed without access to the source code of the application. DAST tests security by replicating attacks against the running application, deliberately looking for runtime vulnerabilities, such as authentication errors, input validation issues, and improperly configured security headers. The author also added that DAST tools are especially useful for web applications since client-server interactions can expose security weaknesses that cannot be seen from viewing the code. Common tools for DAST include OWASP ZAP, Burp Suite, Acunetix, and Netsparker.

3. **Software Composition Analysis (SCA):** Modern applications typically depend on open-source libraries and third-party packages that may already contain known vulnerabilities. Ivanova et al. [39] explains SCA as a tool that identifies and manages related risks by scanning dependencies used in the project and flagging known security issues. It can also provide remediation guidance and licensing information, which is important for compliance. Popular SCA tools include Snyk, Black Duck, WhiteSource, and OSS Index. These tools integrate into the CI/CD pipeline and continuously monitor newly discovered vulnerabilities in third-party components.
4. **Infrastructure as Code (IaC) Security:** IaC enables developers to write code that defines elements such as networks, servers, and permissions. While the method allows for scalability and consistency, it can create new risks to security if misconfigured. According to Verdet et al. [40], IaC security involves scanning IaC configuration files for misconfigurations such as overly permissive roles, unsecured ports, and hard-coded secrets. Scanning tools like Checkov, TFSec, and Terraform Compliance are automating the identification of misconfigurations prior to provisioning infrastructure based on the IaC. Authors also added that integrating the scanning tools as part of a CI/CD pipeline enables secure infrastructure changes.
5. **Container Security:** Technologies for containerization, namely Docker and Kubernetes, have transformed application deployment processes, but they have also created new security risks. According to Rice [41], Sultan et al. [42], Devi et al. [43], container security is about securing the container images and runtime environments from risks associated with vulnerabilities and misconfiguration. Security for container images and runtime environments includes identifying, scanning, and assessing container images for known CVEs, a Common Vulnerabilities and Exposures Framework. Authors also added that container security includes establishing and enforcing security policies for container runtime environ-

ments, isolating the container environment from the host environment, and utilizing best practices for the container image. There are numerous tools for scanning container images for vulnerabilities, including, but not limited to, Trivy, Clair, and Anchore.

In conclusion, these tools and practices represent the foundation of DevSecOps implementation. Direct integration of these tools and practices in the development pipeline enables continuous security without compromising the development lifecycle. Each practice contributes in a distinct manner to identify and manage risk, including identifying it at the code level, application behavior, third-party dependencies, infrastructure, and finally runtime environments to create a robust and resilient software delivery process.

2.3.3 Stages of DevSecOps Lifecycle

The DevSecOps lifecycle is an extension of the traditional DevOps model by embedding security directly into the software development and delivery process, rather than treating it as a separate or different process. The continuous and integrated DevSecOps process proactively identifies, assesses, and mitigates security risks as early as possible throughout the development lifecycle. Importantly, it does this while still maintaining speed and agility in the delivery process. According to Green [44] and Hsu [45], DevSecOps lifecycle typically encompasses a series of stages —plan, code, build, test, release, deploy, and operate— with security practices tailored to each stage. The following paragraphs summarize authors explanation of these stages.

In the planning stage, teams will add security requirements to the project scope by identifying potential threats using risk modeling and compliance analysis. This stage sets the foundation for proactive security by ensuring all stakeholders, including developers, security professionals, and operations teams, are on the same page from the start. During the coding stage, developers will receive their initial coding guidance from secure coding principles. They will also get help from secure coding training and peer reviews. Implementation of SAST and SCA will be early processes to discover any vulnerabilities that may exist in either code or third-party components.

Once the pipeline moves into the build phase, automation is crucial. First, teams use IaC configurations and scan for misconfigurations pre-deployment using tools like Checkov or TFSec. The testing phase includes functional and security testing, such as DAST, to identify vulnerabilities that may occur at runtime. In addition, techniques like data anonymization and data masking are applied to support a controlled testing process. In the release and deployment phases, teams enforce security gates to validate that critical

vulnerabilities are mitigated. In these phases, they perform assessments, compliance checks, and penetration testing to validate production readiness. They will also apply secure deployment practices, such as container image scanning, to avoid deploying to insecure environments.

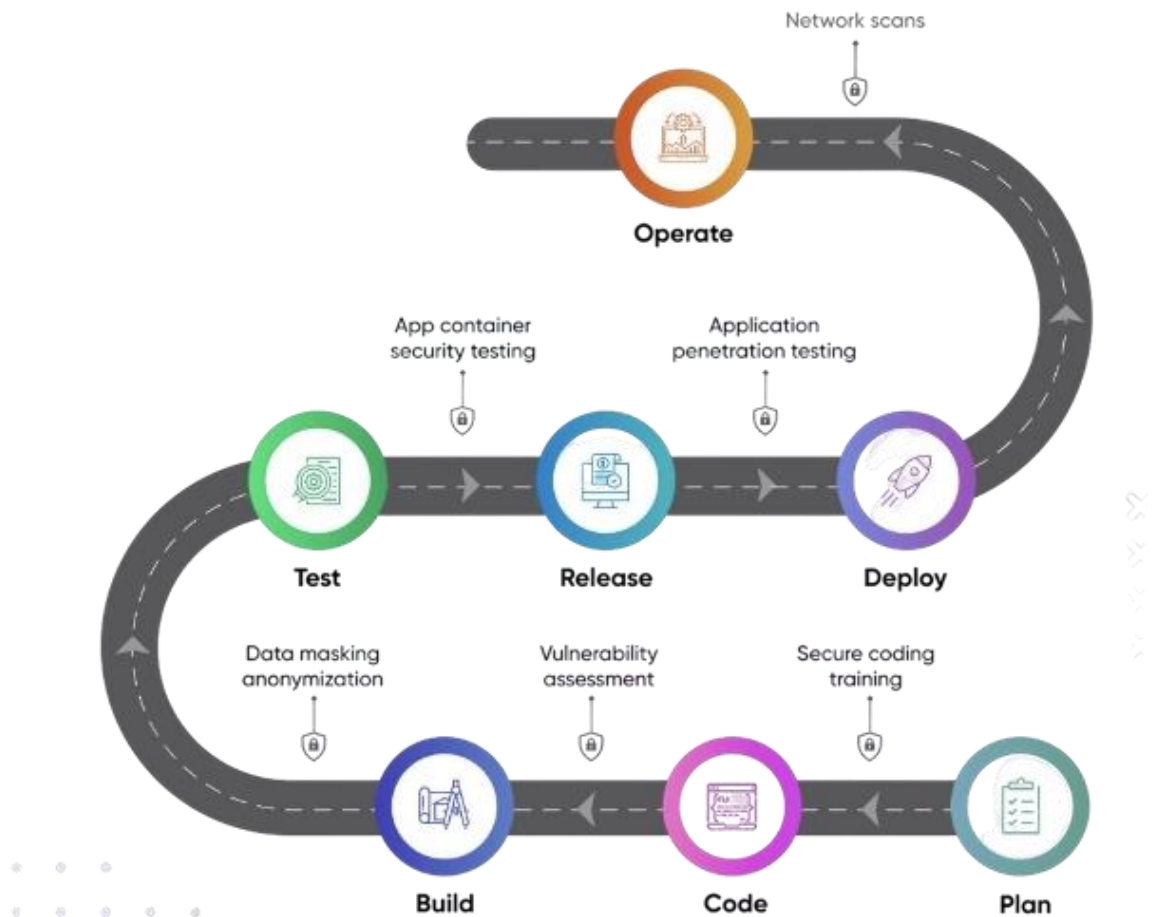


Figure 2.3. Stages of DevSecOps Lifecycle with Security Checkpoints [46]

Figure 2.3 illustrates the stages of DevSecOps containing different phases like code and plan. Following the deployment, the operate stage aims to maintain the security of the application and continuously improve security post-deployment. In this stage, teams perform activities such as runtime application self-protection (RASP), security logging, and continuous monitoring to detect threats and respond to incidents. In addition, network scans, anomaly detection, and audit trails will contribute to a compliant production environment and resilient operations. [44-45]

2.4 Common Vulnerabilities: OWASP Top 10

The Open Worldwide Application Security Project (OWASP) is a nonprofit organization dedicated to improving security solutions for software around the world. One of the

OWASP's key contribution is the OWASP Top 10 [47] —a regularly updated list of the most significant security threats to web applications. This list of threats reflects the opinions of security experts all over the world and is intended to help developers and educators understand common threats and prioritize secure designs for their applications. The following paragraphs summarize OWASP explanation of these vulnerabilities.

Broken access control describes vulnerabilities that allow users to bypass the rules of authorization and gain access to resources that are outside of their permissions. **Cryptographic failures** arise when sensitive data is exposed because of weak or incorrectly used encryption techniques. **Injection vulnerabilities** refer to the act of attackers injecting various forms of malicious code (e.g., SQL commands or OS commands) into applications to manipulate their backend or management systems. **Insecure design** refers to the broader perspective of a system architectural defect, which, unlike coding defects, may involve a fundamental flaw that will not be easily "fixed" with coding alone.

Security misconfiguration describes the situation where either servers, frameworks, or components have been set up incorrectly and left exposed to attack. **Vulnerable and outdated components** refer to when software or applications rely on outdated libraries that have known vulnerabilities. **Identification and authentication failures** describe the weakness in credentials or faulty authentication mechanisms that allow for unauthorized access. **Software and data integrity failures** describe a failure to adequately check integrity for updates and data pipelines, which may also result in some form of unauthorized access.

Security logging and monitoring failures will delay the discovery of the presence of a threat and leave the attacker undetected, and finally, **Server-Side Request Forgery (SSRF)** will occur when a server is fooled into making requests to an unintended destination, such as to an internal network. In DevSecOps, understanding these vulnerabilities is important as many of its tools and practices are designed to detect and mitigate these risks. [47]

2.5 Importance of DevSecOps in Education

As software systems become more complex and tightly coupled, it will become more urgent to design and implement systems securely. According to Nadeem [82], even as we move toward an environment and society that are ever more dependent on software goods and services, computer science and software engineering curricula still view security as something isolated from any thoughtful practice of development. He also added

that often, curricula only introduce security more comprehensively in option domains or later within programs [82].

Therefore, there will be a gap between what universities teach and what employers want. And this is very problematic when modern development viewed as a constant, rapid cycle of delivery through collaboration, such as DevOps, where security can be overlooked. It is clear that security is already viewed in less orthodoxy since it is being approached as a key element in the "DevSecOps" context. DevSecOps integrates security into the development and deployment pipeline rather than simply adding it at the end—when it can be the easiest to avoid. Industry, and employers in particular, are now more commonly expecting developers to implement security methodologies, not just security specialists. However, career surveys note continuing issues for graduates entering the workforce and lacking practical lessons in secure coding, automated testing tools, or creating automatic security testing within new CI/CD pipelines.

The GitLab 2024 Global DevSecOps Report [48] highlights that organizations are increasingly investing in security, AI, and automation, emphasizing the need to implement educational reform to ensure teams are adequately trained to integrate security into development processes. A number of academic initiatives have already shown the benefits of a hands-on, integrated security education. For example, OWASP's University Challenge [49] and other education-based initiatives have encouraged educational institutions to integrate security components into their curricula and activities. Furthermore, the ACM/IEEE Joint Task Force [50] on Computing Curricula has recognized the need for security to be viewed as a cross-cutting concern in software engineering education.

In this context, this thesis aims to contribute to the emerging area of secure software education by designing and conducting a hands-on DevSecOps exercise, with the goal of assessing several variables like knowledge, interest, self-efficacy, and attitude before and after the exercise.

2.6 Theoretical Background of Assessment Variables

To evaluate the impact of the DevSecOps exercise, this study adopts a theoretically grounded set of assessment variables that reflect both cognitive and motivational dimensions of learning. The variables—knowledge, interest, self-efficacy, attitude, and expectations/outcomes—are informed by well-established models such as the Knowledge-Attitude-Behavior (KAB) framework by Kruger & Kearney [92] and social cognitive theory by Bandura [85]. These have been widely applied in the context of cybersecurity educa-

tion and were notably integrated in the work of Schafeitel-Tähtinen et al. [79], who proposed a detailed measurement model to evaluate the effectiveness of teaching interventions.

Their model, tested across five cybersecurity courses and exercises. Schafeitel-Tähtinen et al. [79] identified a set of interconnected variables that influence how students perceive and engage with cybersecurity learning. Empirical findings from their study show that teaching interventions had statistically significant effects on knowledge, skills, cybersecurity-specific self-efficacy, and content-specific self-efficacy. However, student interest levels did not show statistically significant increases post-intervention, suggesting that interest may be more stable or pre-existing rather than a direct outcome of short-term teaching efforts. Their model also highlights that self-efficacy plays a central role in shaping security behavior intentions. Thus, reinforcing the importance of building student confidence through targeted interventions.

Knowledge includes both factual (declarative) and practical (procedural) understanding. In cybersecurity, this spans concepts like threat modeling, secure software practices, and CI/CD integration of security tools. Schafeitel-Tähtinen et al. [79] found that content knowledge significantly increased post-teaching and correlated strongly with perceived self-efficacy. This aligns with the findings of Bell et al. [86], who argue that knowledge acquisition is essential for enabling deeper cognitive engagement in cybersecurity tasks. In the context of DevSecOps, this variable refers to an understanding of security principles, tools, and practices that are applied throughout the software development lifecycle.

Interest refers to a student's emotional and motivational involvement in secure coding and cybersecurity topics. It influences how willing a student is to invest time and effort in learning and applying security practices. According to Hidi [90], interest is a motivational construct that influences how learners allocate attention and effort. Although interest is often associated with better academic performance, its development may depend on longer-term or more personalized teaching strategies. In the study by Schafeitel-Tähtinen et al. [79], interest was found to correlate with knowledge, skills, and attitude but did not significantly increase post-intervention. This observation is echoed in the work of Flowerday and Shell [89], who noted that meaningful learning experiences enhance engagement only when students already find the content personally relevant.

Self-efficacy refers to students' belief in their ability to perform specific tasks, according to Bandura [84]. In cybersecurity education, higher levels of self-efficacy are linked to stronger motivation and persistence, especially in unfamiliar or complex domains. The

model by Schafeitel-Tähtinen et al. [79] distinguishes between general self-efficacy, cybersecurity-specific self-efficacy, and content-specific self-efficacy, and reports significant post-intervention gains in the latter two. Similarly, Colin et al [95] and Bishop et al. [87] emphasized the role of task-specific self-efficacy in predicting students' intent to adopt security behaviors and pursue careers in cybersecurity. In relation to DevSecOps, self-efficacy can determine how students approach security tasks, their persistence when faced with challenges, and their overall efficacy. Research by Mumtaz et al. [51] and Larios-Vargas et al. [52] indicates higher self-efficacy has been shown to increase the likelihood of successful adoption of security practices and tools.

Attitude represents the mindset and predisposition of the learner when integrating security into their workflows. As proposed in Ajzen's [83] Theory of Planned Behavior, attitude influences both intention and action. Schafeitel-Tähtinen et al. [79] observed that while attitude correlated with behavior intention and recalled security actions, it did not show significant change post-teaching. This may reflect that attitude change requires either prolonged exposure or value-based reflection, as also indicated by Faklaris et al. [88] in their SA-6 scale development for measuring end-user security attitudes. When adopting DevSecOps practices, having positive attitudes toward security is a critical factor for success, influencing both motivation and the likelihood of continually implementing security measures.

Expectations/Outcomes refers to what learners expect to gain from their exercise experience, contrasting this with what was actually achieved. This includes expectations for specific skills, knowledge, and competencies, and the extent to which these expectations are met, which also contributes to satisfaction and perceived value. While not measured as a separate construct in Schafeitel-Tähtinen et al. [79], the model incorporates students perceived satisfaction and motivation through the ARCS framework by Keller [91]. According to Loorbach et al. [93] and Pan [94], perceptions of value, alignment with learning goals, and task relevance are known to influence persistence and long-term learning outcomes.

In summary, the selected assessment variables reflect a balance between measurable learning outcomes and motivational-environmental factors. Their inclusion is well supported by prior literature in cybersecurity and educational psychology. Together, these variables provide a multidimensional view of how students experience and benefit from an exercise focused on DevSecOps.

2.7 Linking Assessment Variables to Survey Design

The assessment variables described earlier were translated directly into survey questions (Appendix A). This design process involved explicitly connecting theoretical concepts with measurable items, guided by existing literature from cybersecurity education and educational psychology.

The assessment of **knowledge** was informed by the conceptual distinction between declarative knowledge and procedural knowledge, as elaborated by Bell et al. [86] and Schafeitel-Tähtinen et al. [79]. Accordingly, Section B of the survey includes questions such as "*Static analysis tools (SAST), such as SonarQube, to identify code vulnerabilities*" and "*Building and deploying containerized applications using Docker.*" These items explicitly align with the theoretical definition by evaluating students perceived understanding and familiarity with both conceptual and applied elements of secure software practices in DevSecOps.

The variable **interest**, based on Hidi's [90] theory, was operationalized by crafting items that directly replicate knowledge items but shift the focus from cognitive understanding to affective engagement and motivation. Section C illustrates this strategy, including questions such as "*How interested are you in using Dynamic testing (DAST) tools (e.g., OWASP ZAP)?*" and "*How interested are you in building and deploying containerized applications using Docker?*" This direct correspondence to knowledge items allows the survey to systematically measure motivational engagement in relation to participants perceived knowledge, thereby reflecting the theoretical position that interest influences cognitive involvement and learning persistence.

Drawing from Bandura's [84] social cognitive theory, the variable **self-efficacy** emphasizes individuals' task-specific confidence, a factor strongly predictive of motivation and actual task performance. Section D operationalizes this concept with items explicitly addressing specific DevSecOps tasks, such as "*Integrating security practices into an automated CI/CD pipeline*" and "*Using container security scanning to secure deployment environments.*" These items align closely with Bandura's original conceptualization of self-efficacy as context- and task-dependent, further validated by cybersecurity-specific applications such as Schafeitel-Tähtinen et al. [79]. Thus, the survey instrument assesses confidence rather than knowledge or attitude alone, accurately reflecting self-efficacy's theoretical role.

The **attitude** variable in this research aligns with Ajzen's [83] Theory of Planned Behavior, where attitudes represent evaluative predispositions toward specific behaviors and

significantly shape behavioral intentions. Section E captures this theoretical perspective through evaluative statements such as "*I am motivated to incorporate security practices into my development workflows*" and "*Security should be a shared responsibility among developers, operations, and security teams.*" These items reflect the theoretical claim that positive attitudinal orientation increases the likelihood of actual behavioral engagement, which directly corresponds with Ajzen's model of intention formation.

Lastly, the **expectations/outcomes** variable was assessed by matching pre-survey expectations with post-survey reflections on perceived learning achievements, informed by prior research examining expectation fulfillment and perceived value in educational contexts [93, 94]. Specifically, Section F includes paired items such as the pre-survey statement "*I expect that this exercise will improve my technical skills in implementing secure CI/CD pipelines*" and its post-survey counterpart "*Participating in this exercise has improved my technical skills in implementing secure CI/CD pipelines.*" These paired statements enable an explicit evaluation of participants' initial expectations compared with their subsequent experiences. Thus, these statements align with theoretical perspectives that emphasize perceived alignment between anticipated and realized outcomes as a crucial indicator of instructional effectiveness.

By explicitly deriving survey items from clearly articulated theoretical constructs and ensuring their alignment with prior literature, the survey questions (Appendix A) provide a robust foundation for evaluating the educational impact of the DevSecOps exercise.

3. RELATED WORK

The purpose of this chapter is to explore what others have already done in the areas of DevSecOps and software security education. While earlier parts of the thesis laid out the motivation and theoretical context behind secure software practices, this chapter dives into the work that researchers have contributed so far. It starts by looking at how DevSecOps has been applied in development environments. Then, it turns to how software security is being taught, especially the different methods and tools used to make learning more effective. Finally, the chapter focuses on how DevSecOps itself is being introduced in software security education. Together, these sections help build a clearer picture of what's already known and what gaps remain.

3.1 Search Process

To explore previous approaches in software security and DevSecOps education, a literature search was performed. The literature was searched from well-known databases such as IEEE Xplore, SpringerLink, Web of Science, ScienceDirect, and ACM Digital Library, using the search string outlined in Table 3.1. Only relevant works published between 2015 and 2025 were included in the consideration. In the context of DevSecOps and software security education, there is an abundance of relevant research articles. However, when focusing specifically on DevSecOps within software security education, the author of this thesis found a limited number of relevant works.

Table 3.1. Search strings for relevant work

Subsection	Search String
DevSecOps and Security	("DevSecOps" OR "Development Security Operations" OR "Secure DevOps" OR "DevOps Security") AND (practices OR tools OR adoption OR challenges OR implementation OR education OR training OR "software security" OR "secure software development")
Approaches in Software Security Education	("Software Security Education" OR "Secure Software Development Education" OR "Secure Coding Education" OR "Security Training in Software Development" OR "Integrating Security into Software Development Curriculum" OR "Software Development Security Pedagogy")

DevSecOps in Secure Software Development Education	("Software Security Education" OR "Secure Software Development Education" OR "Secure Coding Education" OR "Security Training in Software Development" OR "Security Education") AND ("DevSecOps" OR "Development Security Operations" OR "DevOps Security" OR "Secure DevOps")
--	---

3.2 DevSecOps and Security

Over the last few years, there's been a growing shift in how software development teams approach security. In the past, security was often an afterthought. It was something added later, maybe right before deployment but things are changing. With the rise of DevSecOps, security is being pushed much earlier into the development cycle. It is becoming part of the entire process, not just a final checkpoint. This shift in mindset has sparked a wave of academic and industry efforts to embed security throughout the development lifecycle. The following studies illustrate how DevSecOps is being implemented across different contexts.

One study by Kumar and Goyal [53] introduced the ADOC model, which is an effort to formalize continuous security within open-source, cloud-based workflows. Their framework proposed structured integration of over 40 security controls at various stages of development. While comprehensive, its success hinges on aligning tools with organizational culture, a factor that can be difficult to standardize. In another context, Marandi et al. [54] experimented with integrating tools like Snyk and StackHawk directly into the development pipeline. These tools perform static and dynamic security testing. What's interesting here is the way they focused on combining both types of tests to catch vulnerabilities more reliably. Their results make it clear that automation does not just save time. It leads to better coverage.

Now, when it comes to low-code development, things get trickier. Sedrakyan et al. [55] investigated how non-technical users, often called citizen developers, might introduce risks without realizing it. They created something called LowDevSecOps, which is basically a set of smart prompts and risk warnings built into low-code platforms. This area is important because, as more companies allow non-developers to build apps, it remains essential for those apps to be secure. Saurabh and Kumar [56] took a more technical angle. They looked at how performing static code analysis early using tools like SonarQube can reduce build times and fix bugs sooner. Their shift-left approach is not exactly new, but their results indicated that making security checks earlier really does make a difference.

Some researchers focused on more specific cases. For example, Drane et al. [57] looked at scientific web apps, especially those built using the single-page application model. They designed a way to manage things like content security policies and Docker deployment rules that help align with DevSecOps. It's very hands-on and practical research, showing how DevSecOps works in real environments. On a broader level, Ramaj et al. [58] studied DevSecOps within critical infrastructure environment. Their research was more qualitative. It is based on expert interviews, but it reinforced the idea that proactive security and monitoring help reduce long-term risks. It's not just about stopping attacks. It's about building systems that can recover and continue operating even when something goes wrong.

Other works, like the one by Casola et al. [59], focused more on automation. They developed a method that links software models to security test plans, enabling developers to test systems early and with fewer manual steps. It's a good example of how threat modeling is slowly becoming part of day-to-day DevOps. Singh et al. [60] explored deeper into automation, demonstrating how automated tools can secure infrastructure code such as Terraform and static security testing. It's clear from these studies that automation is not just a convenience. It's the only way to keep up with modern development speed.

In summary, while these papers focus on different tools and environments, they share the same goal, which is making security automatic, collaborative, and early. DevSecOps is not one thing. It's a mindset shift. But like any cultural change, adoption takes time, and it often depends more on people than tools.

3.3 Approaches in Software Security Education

As software becomes more deeply embedded in critical systems, the importance of teaching security at the foundational level has grown. Yet, educators have long struggled with how to deliver this content perfectly, especially in ways that feel relevant and impactful to students. In recent years, there's been a noticeable shift away from security as an isolated topic and toward more integrated, experience-based methods.

Liu and Ezenwoye [61], for example, describe their experiences with the SecureED platform, which is an intentionally vulnerable web application designed for educational purposes. Within their findings, students explored vulnerabilities, created a threat model, and offered mitigations. They contend that this experiential method supported students in connecting abstract ideas with concrete approaches for secure software design. A similar framework was presented by Shahriar et al. [62], who created and called their

model PASS Labware. The PASS Labware combines pre-lab preparation, hands-on exercises centered around the OWASP Top 10, and reflection after lab activities. Its purpose is to promote not only the gaining of technical skills, but also the retention of long-term understanding, particularly in cross-site scripting and insecure authentication contexts.

Game-based learning is also commonly discussed in the literature. For example, Xie et al. [63] tested secure coding duels on Microsoft's Code Hunt platform, while Zouahi [64] reported on the use of CTF-style competitions to simulate realistic security challenges. While the two studies had different formats, they both reported increased student engagement and retention. Timing is another recurring theme. Taylor and Kaza's [65] "Security Injections@Towson" project embedded modular security lessons directly into early programming courses. Their goal was to instill good habits before students became accustomed to insecure practices, a goal supported by their multi-institutional assessment results.

A different and equally interesting perspective is presented by Nocera et al. [66], who utilized SonarCloud in a course that focused on development of web applications. While students were not explicitly required to fix detected vulnerabilities, many chose to do so. This study suggests that simply integrating such tools into the workflow can encourage students to adopt secure practices, especially when they see the tangible benefits. For learners in part-time or professional programs, other adjustments have proven necessary. Teiniker et al. [67] tailored their course for part-time learners using inductive learning strategies, where students were encouraged to explore and draw conclusions from security case studies. The approach seemed to suit those balancing coursework with professional obligations.

Lwin et al. [68] offer an example from a front-end web development course where students were introduced to ZAP, a security scanning tool. While the intervention helped reduce some types of vulnerabilities, students indicated it came too late in the term to fully apply without major code changes. Their suggestion was that security must be considered from the very beginning of the project.

Rahman et al. [69] explored the use of automated static analysis tools like cppcheck to enhance software security awareness. Their course design intentionally aligned scanning exercises with lecture content and students indicated this helped them to understand security smells in actual codes. This evidence lends support to the argument that reinforcement across multiple formats—lectures, tools, and labs—is important for reten-

tion. Finally, Tao et al. [70] showcased the merits of teaching security with Android application development. Their strategy was to include flaws into learning applications that students would test and fix in real time. Even within a single term, students exhibited measurable growth in their ability to think critically about secured design.

In conclusion, the studies indicate a consensus in current research that effective security education includes dimensions of early exposure, practical tools, continuous feedback, and contextualization regardless of medium (modular labs, real-world tools, interactive games). The goal ultimately is the same, to help learners not only to be aware of software vulnerabilities but to have the mindset and skills to prevent them.

3.4 DevSecOps in Secure Software Development Education

When reviewing the existing research, it became clear that there were not many studies or educational frameworks that focus specifically on teaching DevSecOps principles in the classroom. This was quite surprising given how important DevSecOps has become in the software industry. One notable work is by M. M. Rahman et al. [71], who developed a hands-on learning approach using Git Hooks and automated static security analysis to help students grasp DevOps security concepts. Their study highlights how practical, tool-based learning can build real skills and awareness around security challenges faced in DevOps workflows. Still, beyond this, the academic literature has not quite kept pace with the rapid evolution of DevSecOps practices in real-world development. They have not adopted the necessary practices of DevSecOps.

This lack of focused research indicates to a major gap in software security education. University programs tend to emphasize traditional secure coding or isolated DevOps practices without offering a fully integrated view that combines development, security, and operations into one continuous process. As a result, many students may graduate without the mindset and skills needed to handle the automated, collaborative security demands of today's software projects.

That's why my thesis, "Design and Assessment of DevSecOps Exercise on Secure Software Development Education," is timely and necessary. The goal is to create educational exercises that engage students in real-world DevSecOps scenarios, helping them not just write secure code but also work effectively within modern DevSecOps pipelines.

4. EXERCISE DESIGN AND IMPLEMENTATION

This chapter explains how exercise was designed and implemented. It starts by outlining the key steps involved in its design, providing insight into the process behind its structure. Following this, the chapter reviews the outputs produced by the secure pipeline and highlights the security flaws identified in the test application. The exercise is based on DevSecOps principles and is meant to help participants improve their knowledge and confidence in using security tools during the software development process.

4.1 Methodology

The exercise was designed to provide the participants with hands-on experience of developing a secure Continuous Integration/Continuous Deployment (CI/CD) pipeline by integrating several security tools to simulate a real DevSecOps environment. Figure 4.1 illustrates the overview of the pipeline architecture containing different security checkpoints. To begin with, a virtual machine environment was set up with an Ubuntu Server as the host for the CI/CD pipeline.

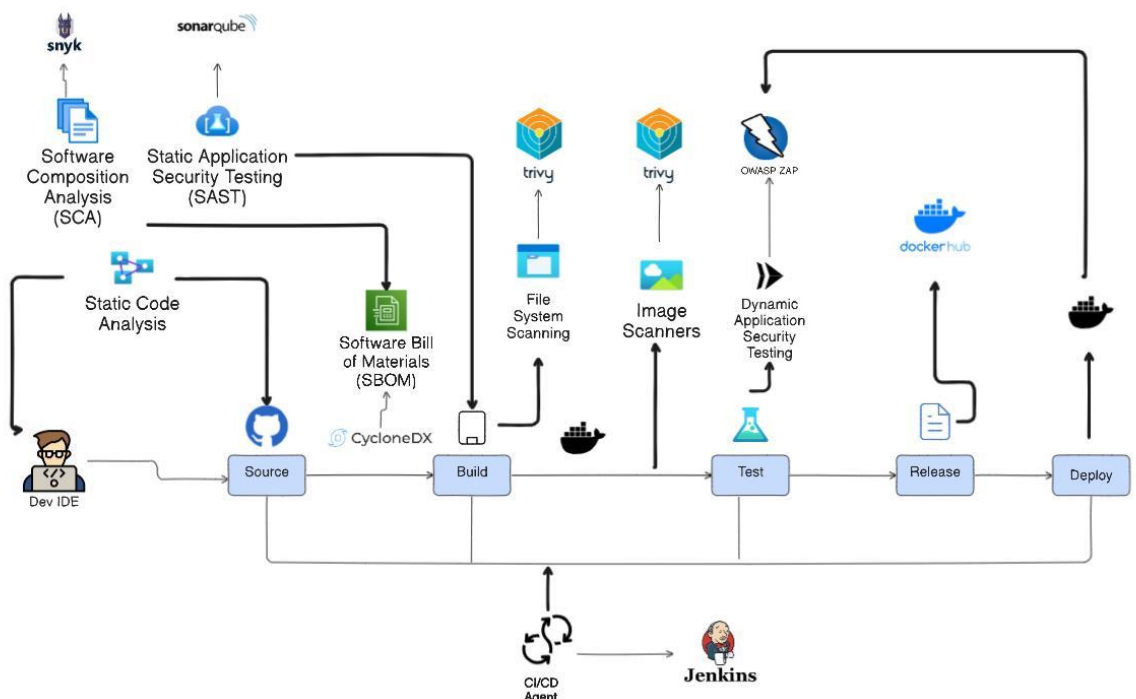


Figure 4.1. Overview of the pipeline architecture

The OWASP Juice Shop application was chosen as the target application [72]. The application was selected since it is open source, and its deliberately vulnerable design makes it ideal for security testing and learning. Jenkins serves as the automation server

for setting up the pipeline because it supports multiple stages of code integration, testing, and deployment. We used Git to clone the OWASP Juice Shop application and manage version control. Then various security testing tools were installed on the host system. The pipeline was configured using these tools. SonarQube was utilized for static application security testing (SAST) to inspect the source code for possible vulnerabilities. We have used Synk as a Software Composition Analysis (SCA) tool to identify vulnerabilities in the third-party dependencies. CycloneDX was also employed to generate a Software Bill of Material (SBOM), which is a report with a list of all components utilized by the application. To verify file system integrity, we scanned the application's file system for security vulnerabilities with Trivy. We then created the test application inside a Docker image and ran a Trivy image scan to find any potential vulnerabilities. The Docker image was then pushed into DockerHub and run as a container inside the host system. We performed dynamic application security testing (DAST) using the OWASP ZAP tool. It runs a baseline scan by simulating attacks on the running application to detect vulnerabilities. Finally, we used the generated artifacts from all these scans to find and report the test application's vulnerabilities. This process not only helped to prioritize the vulnerabilities based on their severity but also allowed to make informed decisions about necessary remediation steps.

4.2 Analysis of Exercise Outputs

The practical aspects of this exercise involved creating a secure CI/CD pipeline using Jenkins and various DevSecOps tools. The pipeline specifically targeted the OWASP Juice Shop application, which is widely recognized for its numerous vulnerabilities [72]. It is also frequently used in educational security testing. This section explains outputs from different stages of the pipeline as well as provides a thorough analysis of what the participants experienced.

4.2.1 Overview of CI/CD Pipeline

After successful execution of all the required steps of the exercise, participants were able to get this dashboard in Jenkins. This dashboard provides basic information about this pipeline, such as build artifacts, user, and execution time. In Figure 4.2, you can view the build artifacts, including scan reports from various security checks. One can also navigate to other options, such as pipeline overview and pipeline steps, from the dashboard. Figure 4.3 presents the pipeline overview, through which one can view the pipeline steps. Green ticks represent successful execution of that step.

Jenkins Dashboard > SecurePipeline > #4

Status: ✔ #4 (Apr 17, 2025, 8:21:48 PM)

Build Artifacts:

Artifact	Size	Action
Container_Scan_Report_Trivy.html	173.25 KIB	view
DAST_Report.html	166.94 KIB	view
File_System_Scan_Report_Trivy.json	14.01 KIB	view
sbom.json	4.07 MB	view

Started by user Musfiqur Milton

This run spent:

- 12 ms waiting;
- 54 min build duration;
- 54 min total from scheduled to completion.

git Revision: 5805125c44098d61425a756a915e79f7400efb05
Repository: <https://github.com/musfiqur-mjuice-shop.git>

Warning: The following steps that have been detected may have insecure interpolation of sensitive variables (click here for an explanation):

- sh: [SNYK_TOKEN]

No changes.

Figure 4.2. Pipeline Dashboard

Jenkins Dashboard > SecurePipeline > Stages

Build SecurePipeline

Build | Configure

pipeline

Start | Tool/Install | Clean Workspace | Checkout from ... | SAST - SonarQ... | Generate SBO... | SCA - Snyk | File System Sca... | Docker Build & ... | Container Scan... | Deploy Container | DAST - OWASP... | Post Actions | End

#4

Figure 4.3. Pipeline Overview with all stages

In practice, the Jenkins dashboard and pipeline overview serve not only as visual confirmation of successful completion but also as diagnostic tools. Participants could utilize these views to quickly identify failed steps or bottlenecks, aiding effective troubleshooting during the exercise.

4.2.2 Static Analysis

SonarQube scanned the entire source code of the OWASP Juice Shop application and provided a dashboard view of its findings. It identified various types of issues, including bugs, vulnerabilities, and code smells. The severity of these issues was categorized as critical, major, minor, and informational. The application has approximately 21 bugs and 269 code smells, with 41 classified as critical severity, 160 as major severity, 68 as minor severity, and 21 as informational severity. Figure 4.4 and 4.5 illustrates results from the security scan. The application has around 232 security hotspots, of which 161 belong to authentication, as it used hardcoded credentials. Furthermore, there are some high-severity issues, such as insecure JavaScript functions and unsanitized user inputs, that can lead to code injection or cross-site scripting. The application had some security

hotspots with denial of service, permission, weak cryptography, and insecure configurations. For each of these issues, SonarQube provided information about the risk as well as how it can be fixed.

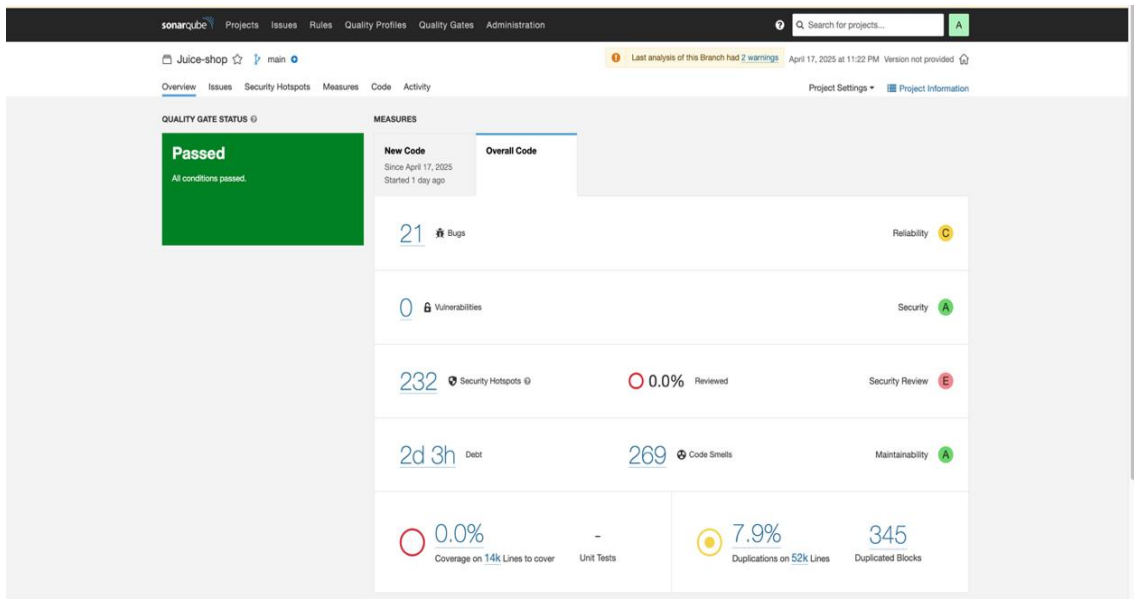


Figure 4.4. Dashboard of SonarQube Scan

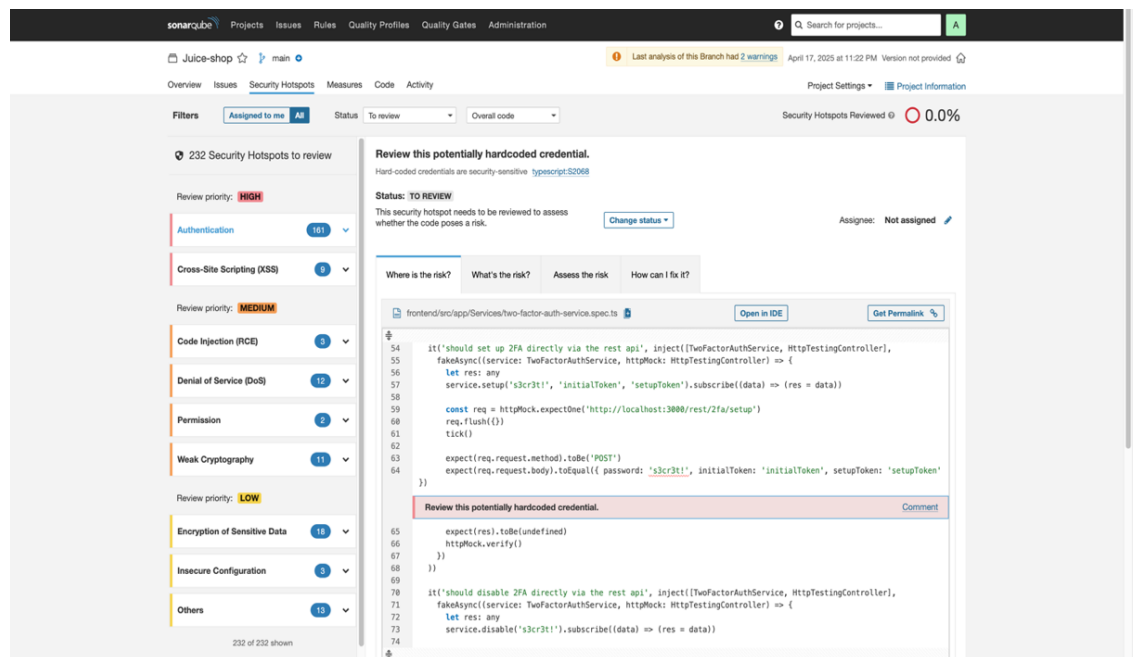


Figure 4.5. Detailed view of Security Hotspots

These findings provided participants with a hands-on understanding of how static analysis tools, such as SonarQube, contribute to secure software development. By reviewing issues and suggested remediations, learners could reflect on how even minor code anomalies may escalate into serious security risks if left unaddressed.

4.2.3 Software Composition Analysis

Synk application provided insights into vulnerabilities that are present in third-party dependencies in OWASP Juice Shop. It has identified 57 issues, with 5 in critical, 22 in high, 27 in medium, and 3 in low severity levels. Additionally, a priority score aids in identifying the issue that requires immediate resolution. For each of the security issues, if there are any fixes available, Synk provides recommendations for it. Out of 57 issues, 15 have no fixes available.

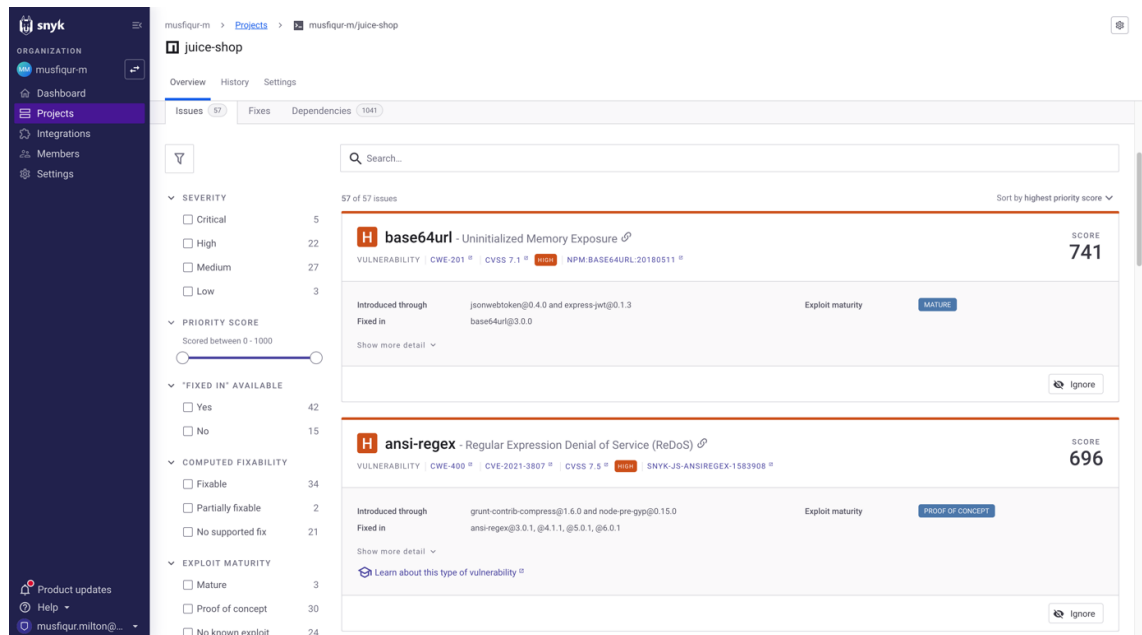


Figure 4.6. Scan Report of Synk

Figure 4.6 represents the results from software composition analysis. The scan results also identified popular packages like `lodash`, which was flagged for multiple high- and medium-severity issues. Most critically vulnerable components, like “`sanitize-html`” and “`vm2`,” were prone to arbitrary code execution, and remote code execution [80]. This tool enabled participants to comprehend the management of security issues arising from third-party dependencies.

4.2.4 File System and Image Scanning

We performed a Trivy file system scan to find sensitive data like exposed tokens or API keys in the source file system. It has identified several security issues in the OWASP Juice Shop. The most critical findings were two hardcoded RSA private keys found in the file directory. The scan also detected security flaws in two files that had JWT tokens directly incorporated into the code. These findings provided participants an example of how easily sensitive information can leak into the codebase. When we ran a container

image scan using Trivy on the generated images, several known vulnerabilities in both the operating system and application layer were identified. Essential system libraries like libc6, openssl, and libssl1.1 also had some security issues. Vulnerabilities such as CVE-2023-4806 and CVE-2023-4813 in libc6 can help attackers leak information or crash processes [81]. In addition to this, the scan also found vulnerabilities in popular JavaScript libraries like crypto-js, jsonwebtoken, and lodash within the application. One critical vulnerability in crypto-js was due to weak cryptography functions (CVE-2023-46233). Figure 4.7 illustrates scan report of container scan which includes different identified vulnerabilities.

**musfiqur97/juice-shop:latest (debian 11.10) - Trivy Report - 2025-04-17 21:10:45.587914249 +0000 UTC
m=+26.411291971**

debian					
Package	Vulnerability ID	Severity	Installed Version	Fixed Version	Links
libc6	CVE-2023-4806	MEDIUM	2.31-13+deb11u10		http://www.openwall.com/lists/oss-security/2023/10/03/4 http://www.openwall.com/lists/oss-security/2023/10/03/5 http://www.openwall.com/lists/oss-security/2023/10/03/6 Toggle more links
libc6	CVE-2023-4813	MEDIUM	2.31-13+deb11u10		http://www.openwall.com/lists/oss-security/2023/10/03/8 https://access.redhat.com/errata/RHSA-2023-5453 https://access.redhat.com/errata/RHSA-2023-5455 Toggle more links
libc6	CVE-2025-0395	MEDIUM	2.31-13+deb11u10		http://www.openwall.com/lists/oss-security/2025/01/22/4 http://www.openwall.com/lists/oss-security/2025/01/23/2 http://www.openwall.com/lists/oss-security/2025/04/13/1 Toggle more links
libc6	CVE-2010-4756	LOW	2.31-13+deb11u10		http://cve.mitre.org/cgi-bin/cve/search.cgi?id=2010-4756 http://securityreason.com/achievement_securityalert/89 http://securityreason.com/achievement_securityalert/92/3 Toggle more links
libc6	CVE-2018-20796	LOW	2.31-13+deb11u10		http://www.securityfocus.com/bid/107160 https://access.redhat.com/security/cve/CVE-2018-20796 https://dibbugn.gnu.org/cgi/bugreport.cgi?bug=34141 Toggle more links
libc6	CVE-2019-1010022	LOW	2.31-13+deb11u10		https://access.redhat.com/security/cve/CVE-2019-1010022 https://nvd.nist.gov/vuln/detail/CVE-2019-1010022 https://security-tracker.debian.org/tracker/CVE-2019-1010022 Toggle more links
libc6	CVE-2019-1010023	LOW	2.31-13+deb11u10		http://www.securityfocus.com/bid/109167 https://access.redhat.com/security/cve/CVE-2019-1010023 https://nvd.nist.gov/vuln/detail/CVE-2019-1010023 Toggle more links
libc6	CVE-2019-1010024	LOW	2.31-13+deb11u10		http://www.securityfocus.com/bid/109162 https://access.redhat.com/security/cve/CVE-2019-1010024 https://nvd.nist.gov/vuln/detail/CVE-2019-1010024 Toggle more links
libc6	CVE-2019-1010025	LOW	2.31-13+deb11u10		https://access.redhat.com/security/cve/CVE-2019-1010025 https://nvd.nist.gov/vuln/detail/CVE-2019-1010025 https://security-tracker.debian.org/tracker/CVE-2019-1010025 Toggle more links

Figure 4.7. Scan Report from Image Scanning

Furthermore, outdated versions of jsonwebtoken had high severity since they could compromise user authentication. These findings explain to participants the need for container image scanning as well as show how vulnerabilities can exist in different layers of an application.

4.2.5 Dynamic Testing

The OWASP ZAP scan of the deployed Juice Shop application discovered several runtime vulnerabilities. We detected a total of 17 alerts, of which 4 were medium, 7 were low, and 6 were informational. One of the important issues was the absence of a content security policy header, which could expose the application to cross-site scripting vulnerabilities. It has also identified a permissive cross-origin policy with "Access-Control-Allow-Origin" set to *. It resulted in unauthorized data transfer. We have also identified the threat of possible session hijacking since it lacks anti-clickjacking headers and includes

session IDs within URLs. Low-severity issues included JavaScript security vulnerabilities, usage of deprecated headers, and disclosure of internal IP addresses.



Sites: <http://cdnjs.cloudflare.com> <http://128.214.254.0:3000>

Generated on Thu, 17 Apr 2025 21:14:23

ZAP Version: 2.16.1

ZAP by [Checkmarx](#)

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	4
Low	7
Informational	6
False Positives:	0

Summary of Sequences

For each step: result (Pass/Fail) - risk (of highest alert(s) for the step, if any).

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	12
Cross-Domain Misconfiguration	Medium	11
Missing Anti-clickjacking Header	Medium	11
Session ID in URL Rewrite	Medium	13
Cross-Domain JavaScript Source File Inclusion	Low	12
Dangerous JS Functions	Low	2
Deprecated Feature Policy Header Set	Low	11
Insufficient Site Isolation Against Spectre Vulnerability	Low	10
Private IP Disclosure	Low	1
Timestamp Disclosure - Unix	Low	5
X-Content-Type-Options Header Missing	Low	12
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	11
Non-Storable Content	Informational	3
Session Management Response Identified	Informational	2
Storable and Cacheable Content	Informational	1
Storable but Non-Cacheable Content	Informational	8

Figure 4.8. Scan Report from Dynamic Testing

Figure 4.8 represents the scan report from the dynamic testing. It highlights the issues that were previously discussed. This outcome provided the participants with a hands-on experience of how dynamic testing can discover vulnerabilities that static analysis misses. Through these findings, they became aware of how security headers, session management, and server settings contribute to the security of a web application.

5. ASSESSMENT DESIGN AND ANALYSIS

This chapter presents the evaluation of the developed exercise. First, we explain how the survey was created to assess the exercise and how data was collected and analyzed. Next, we presented the results of the quantitative and qualitative analysis of data collected through surveys from participants. These findings not only highlight the effectiveness of the CI/CD pipeline in improving application security but also demonstrate its impact as an exercise for participants.

5.1 Methodology

This section outlines the methodology followed in the assessment phase. It includes details on how participants were recruited, how the surveys were designed, how the data was collected and pre-processed, and how the analysis was conducted.

5.1.1 Participant Recruitment

This exercise was conducted as part of the exercise work for the secure programming course [96] at Tampere University in Spring 2025, meaning all students enrolled in that course participated in this research. The exercise was worth 16 points in total: 8 points for the basic implementation part and another 8 points for the advanced implementation, which required students to complete additional tasks. We assigned surveys to participants in the exercise to evaluate its effectiveness. Responding to the survey was mandatory for all the participants in the exercise. We instructed participants to create unique survey codes using the provided information. This approach ensures ethical compliance and maintains anonymity. It also helped to match pre- and post-survey responses without collecting personal information.

5.1.2 Survey Design and Structure

Two comprehensive surveys were implemented to assess knowledge, interest, self-efficacy, and attitude of participants towards DevSecOps practices both before and after the exercise. The objective of these surveys was to understand the backgrounds of the participants, knowledge and interest in key technologies, confidence in DevSecOps operations, attitude towards integration of security in development workflows, and expectations/outcomes from the exercise.

Pre-Survey

The pre-survey included several sections. At first, participants provided their demographic information, such as age, gender, professional role, education level, experience in software development, major of their study, familiarity with CI/CD pipelines, and DevSecOps technologies. Then the survey assessed the existing knowledge and interests of participants in the DevSecOps concept and different security testing tools. They rated their current understanding and interest in various topics, including continuous integration/continuous deployment (CI/CD) pipelines, static application security testing (SAST), software composition analysis (SCA), file system scanning, dynamic application security testing (DAST), and container security scanning. Ratings were given on a Likert scale ranging from “I don’t know this” to “I know this thoroughly” for the case of knowledge assessment. The rating scale for interest ranged from “I am not interested at all” to “I am extremely interested.” In both cases, there was an option to indicate inability to assess their knowledge and interest. Then, in the next section, participants evaluated their self-efficacy in performing different tasks in DevSecOps. Some of those tasks were integrating security into automated CI/CD pipelines, using different security testing tools to detect vulnerabilities, and assessing the security of applications using artifacts from the pipelines. Confidence levels were measured on a scale from “no confidence” to “completely confident.” In the next section, the survey explored the attitude of the participants toward integrating security in the development process. This subsection covers the views on the necessity and effectiveness of security integration, the benefits of early detection, and the advantages of automating the security. The response was measured on a Likert scale ranging from “Strongly Disagree” to “Strongly Agree.” Finally, we asked participants to share their expectations for this exercise. We again measured it on the Likert scale.

Post-Survey

With slight changes in the questions to reflect the experiences the participants acquired from this exercise, the post-survey almost followed the pre-survey structure. Then, the participants reflected on their progress in understanding DevSecOps concepts as well as the relevancy of this exercise to their career goals. We replicated the sections that assessed knowledge and interest in DevSecOps tools, self-efficacy, and attitude towards integrating security in development, just as we did in the pre-survey. This helped us directly compare the improvements resulting from participation in this exercise. The final section of this post-survey was unique because it allowed participants to share their reflections on the learning and experiences gained from this exercise. Participants assessed the effectiveness of this exercise in terms of enhancing their technical skills, self-

efficacy with security testing tools, and offering practical insights into real-world applications. We collected responses using the Likert scale, providing an additional option of "I don't know" for those unable to assess the question's statement. We also included a new section in the post-survey to collect open-ended feedback from the participants. The section included questions about the most helpful part of the exercise, exercise improvement, and key takeaways from this exercise.

5.1.3 Data Collection and Preprocessing

Both pre- and post-surveys were mandatory for the participants of this exercise. There were around 104 responses for the pre-survey and 78 responses for the post-survey. There were more responses in the pre-survey because some participants submitted multiple responses. This was likely due to the instruction requiring them to submit a screenshot of the pre-survey completion message along with their exercise return. Some students failed to capture the screenshot during their subsequent pre-survey attempts. To ensure data integrity, these duplicate responses from the pre-survey were removed in the first stage of data preprocessing [76]. There were also some responses with survey code mismatches between the pre-survey and post-survey. These responses with mismatching survey codes were then excluded in the next stage of preprocessing. The data analysis included a dataset containing 75 complete response pairs.

5.1.4 Data Analysis

Figure 5.1 illustrates the assessment design for this exercise. We developed the survey questions based on the theoretical framework and assessment variables discussed in Chapter 2, particularly in Section 2.7, where the process of translating theoretical constructs into survey items is described in detail. They were categorized into five question groups: knowledge, interest, self-efficacy, attitude, and expectations/outcomes. The pre-survey contained the Expectations group, whereas the post-survey included the Outcomes group. We calculated Cronbach's alpha for each question group to assess the internal consistency of the survey instruments. These groups were subsequently utilized as variables, as explained in the previous chapter of the thesis. A Cronbach's alpha value of 0.7 was considered acceptable for reliability [73]. Then the Shapiro-Wilk test was conducted to determine the normality of data distributions for each question group. As the data did not follow a normal distribution, a non-parametric test was selected for hypothesis testing [75].

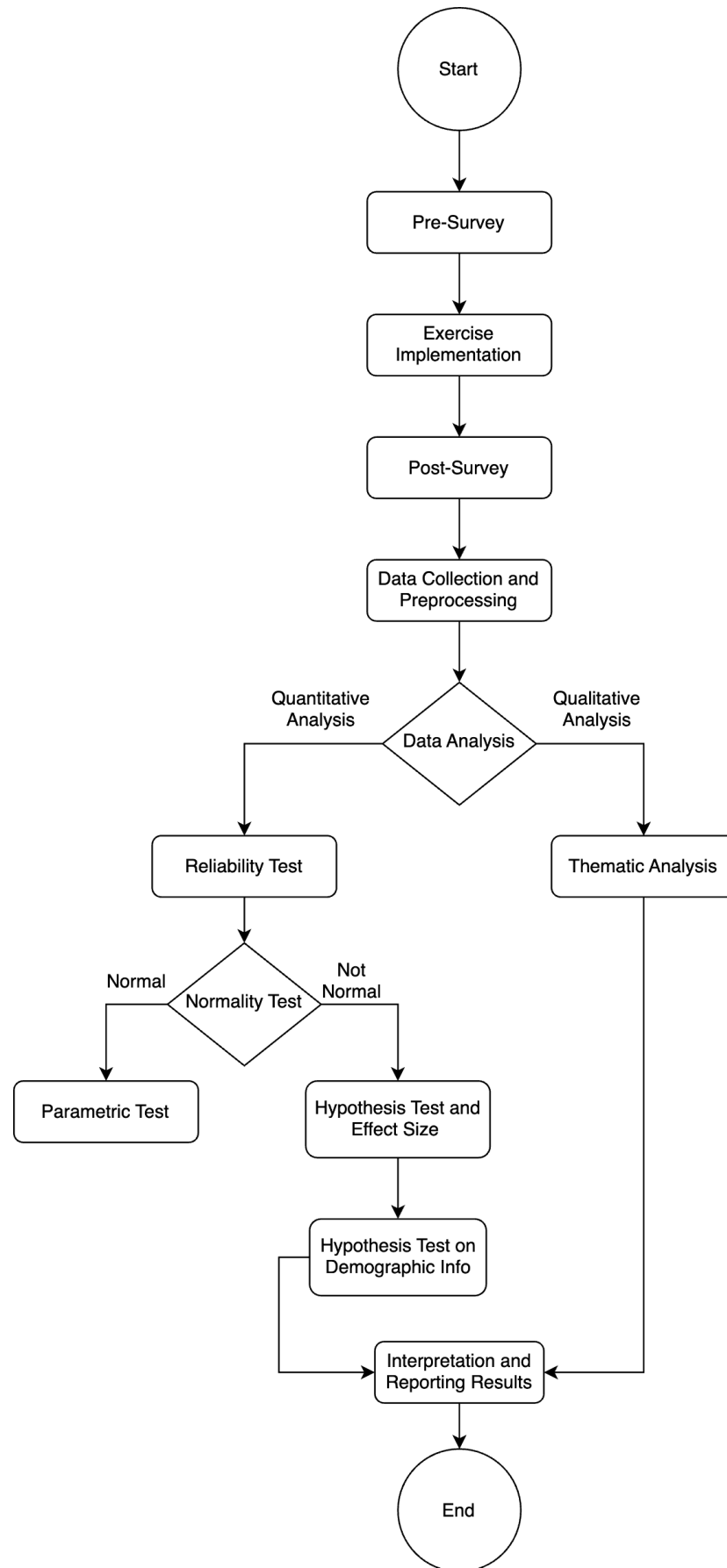


Figure 5.1. Overview of the Assessment Design

The Wilcoxon signed-rank test was used to evaluate the difference in participants responses before and after the exercise. This test is appropriate for paired and non-normally distributed data [77]. It assesses whether the median difference between paired responses is statistically significant. This metric offers insight into the practical significance of the findings, with thresholds interpreted as small ($d = 0.1$), medium ($d = 0.3$), and large ($d = 0.5$) effects [79]. Similarly, we applied the Wilcoxon signed-rank test to the participants' demographic information. Then, we assessed their impact on the variables related to knowledge, interest, self-efficacy, attitude, and expectations/outcomes. All quantitative analyses were conducted using IBM SPSS Statistics 29.0.2 software. Then, we used the responses to three open-ended feedback questions in the post-survey for qualitative analysis. These questions asked about the most helpful part of the exercise, exercise improvement, and key takeaways from this exercise. We coded the responses using MAXQDA24 software. Then, the frequency of these codes was calculated, and finally, themes were generated for each of these questions. This thematic analysis allowed us to identify common patterns and insights in participants' experiences. By combining quantitative and qualitative data, we presented a comprehensive overview of the research and its impact.

5.2 Quantitative Analysis

This section presents the results of the quantitative analysis of the responses from the pre- and post-surveys. We designed the surveys to evaluate the changes in participants' knowledge, interest, self-efficacy, attitudes, and perceived outcomes following the exercise. We will begin by discussing the results of the reliability and normality tests. Next, we will present the findings from hypothesis testing and the calculation of effect sizes. Finally, we will discuss the results of the hypothesis tests based on demographic information.

5.2.1 Reliability Testing

The reliability of the survey instrument was a critical consideration in ensuring the validity of the evaluation. Since each question group was designed to capture a distinct aspect of learner response, internal consistency needed to be confirmed prior to conducting further analysis. Cronbach's alpha was calculated to evaluate the internal consistency for each variable —knowledge, interest, self-efficacy, attitude, and expectations/outcomes. Each variable had alpha values over 0.8, which indicates excellent reliability. Table 5.1 represents the results of the reliability test for both the pre- and post-survey.

Table 5.1. Results of the Reliability Test

Question Group	Pre-Survey	Post-Survey
Knowledge	0.82	0.95
Interest	0.93	0.93
Self-Efficacy	0.94	0.95
Attitude	0.82	0.90
Expectations/Outcomes	0.85	0.93

This ensured that the survey questions for both pre- and post-survey within each group measured the intended concept.

5.2.2 Normality Testing

The Shapiro-Wilk test was applied to determine whether the data followed a normal distribution. The results from table 5.2 showed that the distributions for most variables deviated from normality ($p < 0.05$) [74].

Table 5.2. Results of the Normality Test

Variables	Pre-Survey	Post-Survey
Knowledge, sKN	<0.001	0.075
Interest, sIN	0.422	0.070
Self-Efficacy, sSE	<0.001	0.036
Attitude, sAT	<0.001	<0.001
Expectations/Outcomes, sEM	<0.001	<0.001

Conversely, interest from the pre-survey as well as knowledge and interest from the post-survey show no significant deviation from normality ($p > 0.05$). However, as most of the variables did not meet the assumption of normality, non-parametric methods were deemed appropriate for further statistical analysis [75].

5.2.3 Hypothesis Testing and Effect Size

The Wilcoxon signed-rank test was applied to compare the pre- and post-survey responses of participants to determine whether the exercise had a significant impact on their knowledge, interest, self-efficacy, attitudes, and perceived outcomes.

Impact on Knowledge

We must observe the data distribution in both the pre- and post-survey of the knowledge variable before assessing the impact. Figure 5.2 represents the data distribution of knowledge variable. In figure, sKN_pre and sKN_post represent the knowledge variables

obtained from pre- and post-surveys. The x-axis displays the participants' self-assessments, ranging from 1 to 5 in median values. A rating of 1 signifies "I don't know anything about this," while a rating of 5 indicates "I know this thoroughly." Thus, based on this figure, we can observe a visual improvement in knowledge. To assess whether there is a statistically significant improvement, we employed the related sample Wilcoxon signed-rank test. Figure 5.3 illustrates the differences between pre- and post-responses. So, according to figure, it is evident that almost all responses indicate positive differences.

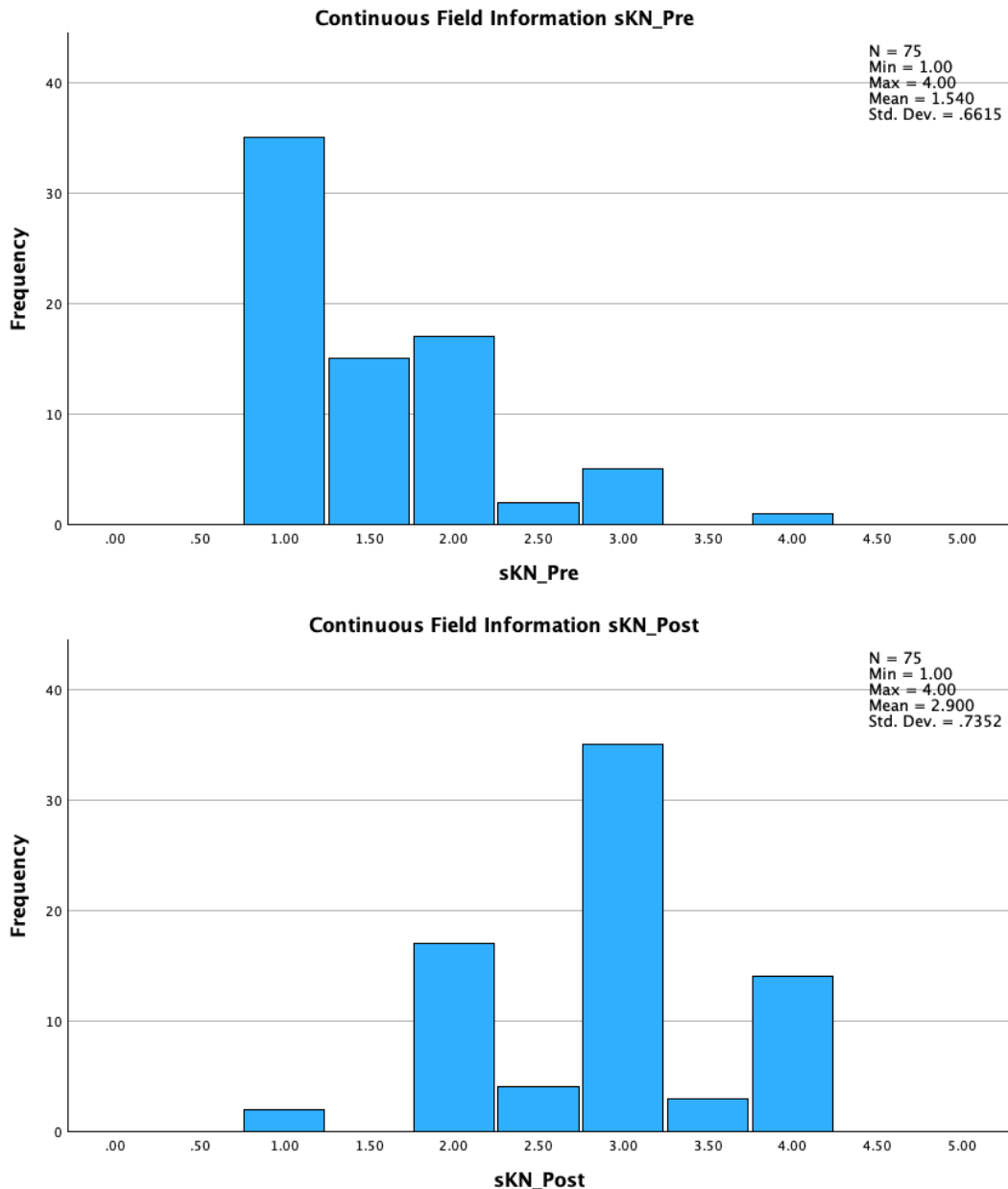


Figure 5.2. Data distribution of Knowledge, sKN variable

There were two instances of negative differences, suggesting that the post-survey ratings were lower than the pre-survey ratings.

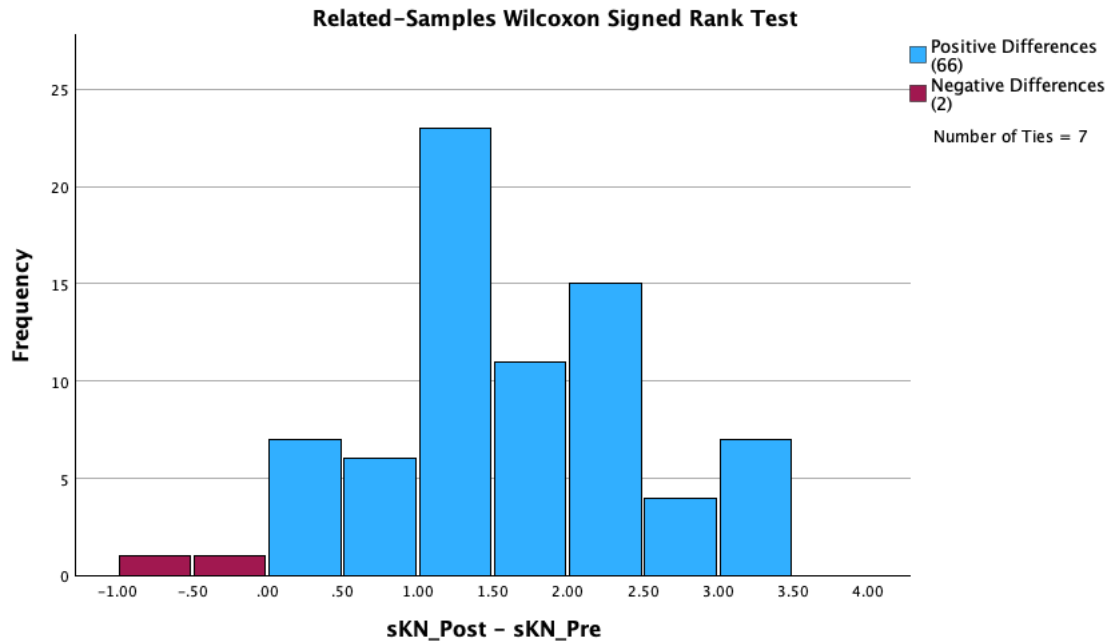


Figure 5.3. Differences between pre- and post-survey responses of sKN variable

These outliers may reflect individual variability in prior expectations or confusion during the exercise. However, given the overwhelmingly positive trend across the dataset, such responses are unlikely to influence the overall interpretation of the knowledge improvement. For seven responses, the ratings were identical. These positive differences resulted in a high Z value of -7.077. Consequently, the test rejected the null hypothesis, as the p-value is less than 0.001. We determined the effect size for knowledge to be 0.8171 by applying Cohen's d, indicating a large effect ($d > 0.50$) [79]. Therefore, with respect to the knowledge variable, we can conclude that there is a statistically significant improvement in knowledge, suggesting that the exercise had a substantial impact on participants' understanding of secure development practices.

Impact on Interest

Before evaluating the impact on interest, we will first analyze the data distribution of the variable sIN in both the pre- and post-surveys. Figure 5.4 illustrates the distribution of data for the interest variable. We represent the interest variables derived from the pre- and post-surveys as sIN_pre and sIN_post. The x-axis illustrates the self-assessments of the participants, which range from 1 to 5 in median values. A rating of 1 indicates "I am not interested at all," whereas a rating of 5 denotes "I am extremely interested." The figure indicates that there were no significant changes in participants' levels of interest. Before beginning the exercise, participants exhibited a high level of interest, which remained consistent throughout. Furthermore, in the post-survey, participants provided slightly higher ratings, with more individuals giving scores of 4 or 5. This stable level of

interest may indicate that the participants already recognized the relevance of secure software practices in their academic or professional development. Maintaining high interest levels, despite the technical complexity of the exercise, demonstrates that the educational content was appropriately designed to resonate with learners' existing motivations.

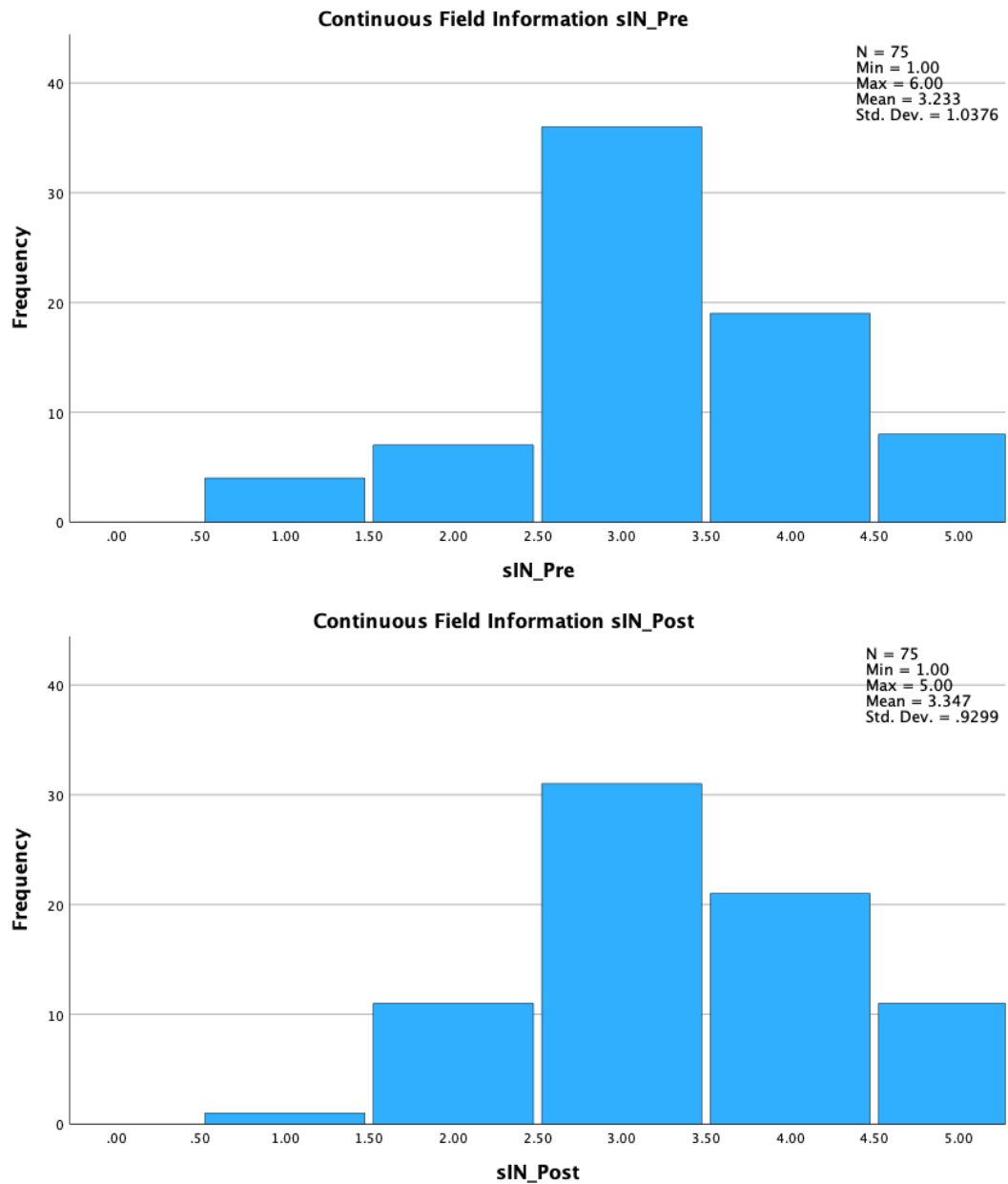


Figure 5.4. Data distribution of Interest, sIN variable

Figure 5.5 illustrates the differences between pre- and post-responses. It shows that most of the responses remained identical. However, there were slightly more positive differences, indicating that there were more instances in which the post-survey rating was higher than the pre-survey rating. Due to these small differences, we ended up with

a low Z value of -0.776. Consequently, the test could not reject the null hypothesis, as the p-value is 0.438. We determined the effect size for interest to be 0.089 by applying Cohen's d, which indicates a marginal effect ($d < 0.1$) [79].

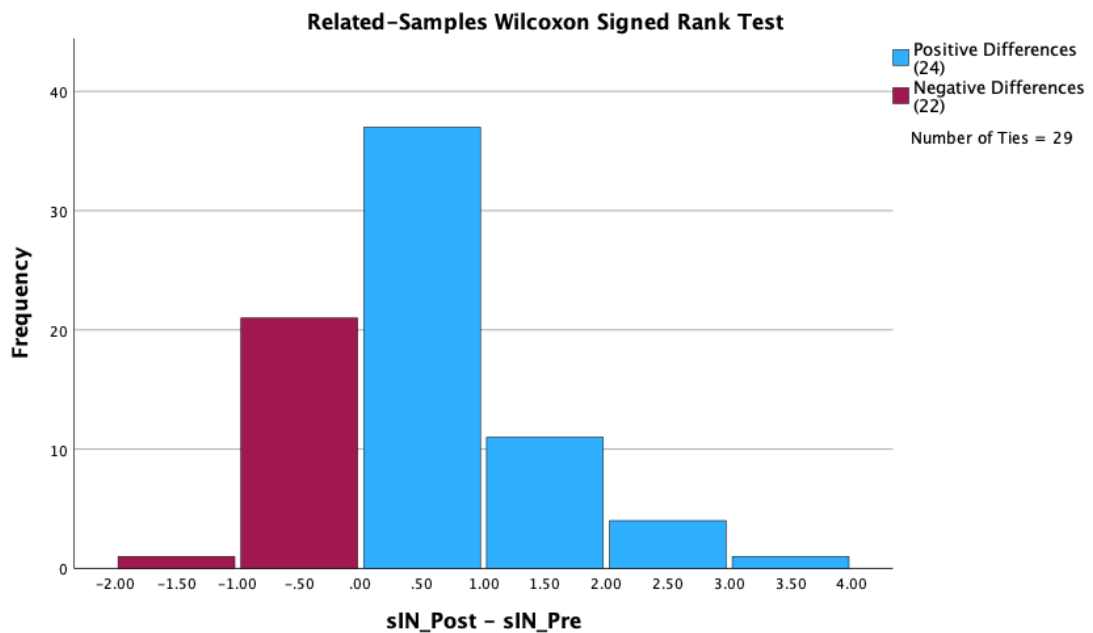


Figure 5.5. Differences between pre- and post-survey responses of sIN variable

Therefore, regarding the interest variable, we can conclude that there is no statistically significant improvement in interest, as the participant already had a higher interest in the topic, and it remained consistent throughout the exercise. This consistency indicates that the exercise effectively emphasized the practical relevance and importance of DevSecOps.

Impact on Self-efficacy

Again, before evaluating the impact on self-efficacy, we will first analyze the data distribution of the variable sSE in both the pre- and post-surveys. Figure 5.6 provides a visual representation of the data distribution for the self-efficacy variable. We represent the variables derived from the pre- and post-surveys as sSE_pre and sSE_post. The x-axis illustrates the self-assessments of the participants, which range from 1 to 5 in median values. A rating of 1 indicates "no confidence at all," whereas a rating of 5 denotes "completely confident." Thus, based on this figure, we can observe a visual improvement in self-efficacy. This increase suggests that the exercise helped participants feel more capable of applying secure development practices in a practical setting. For many, this likely included using the tools with more confidence or understanding their role within the CI/CD process.

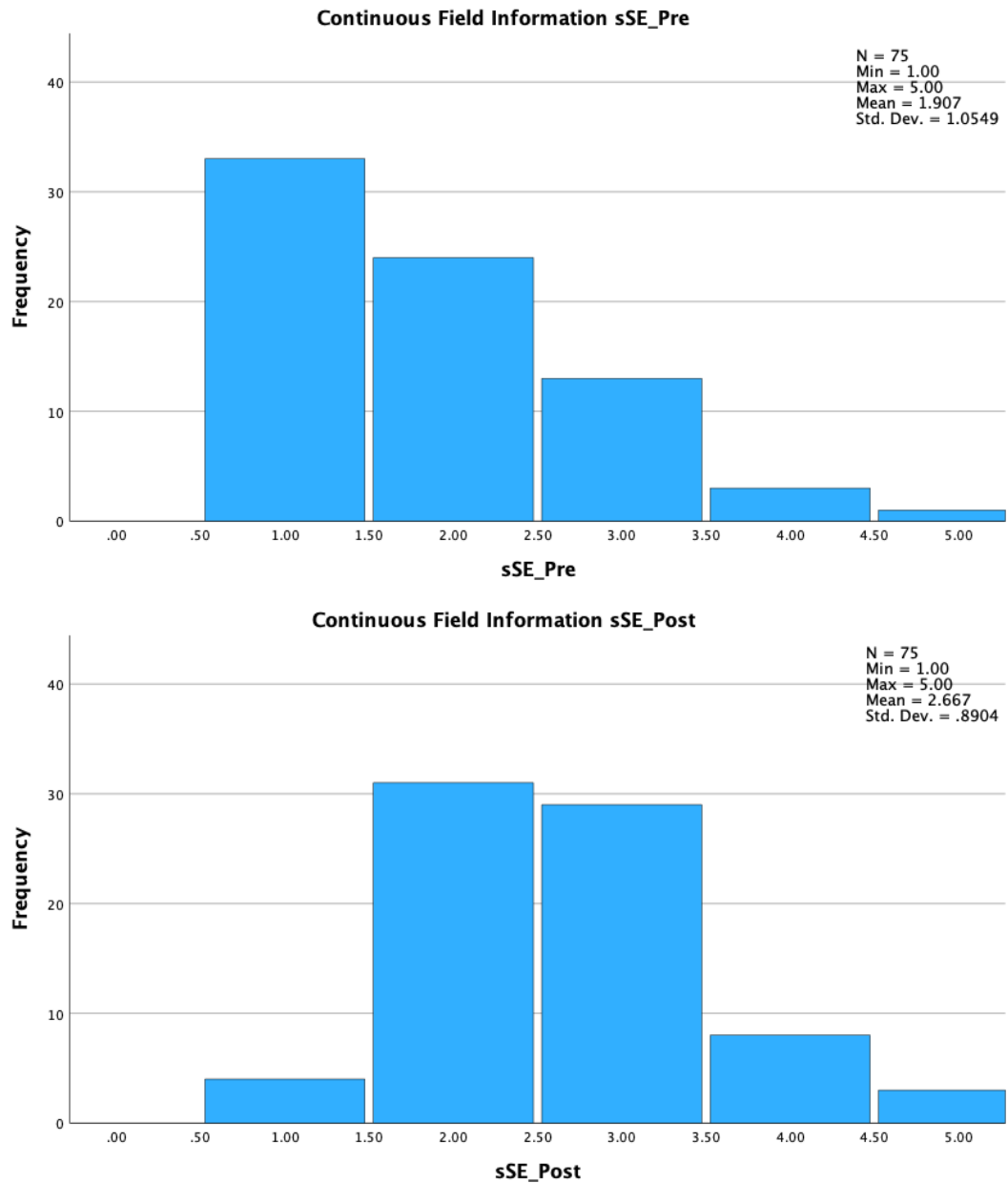


Figure 5.6. Data distribution of Self-efficacy, sSE variable

To assess whether there is a statistically significant improvement, we employed the related sample Wilcoxon signed-rank test. Figure 5.7 illustrates the differences between pre- and post-responses. According to the figure, most responses indicate positive differences. There were five instances of negative differences, suggesting that the post-survey ratings were lower than the pre-survey ratings. For 22 responses, the ratings were identical. These positive differences resulted in a high Z value of -5.433. Consequently, the test rejected the null hypothesis, as the p-value is less than 0.001. The statistical results confirm that the exercise had a significant positive effect on participants' self-efficacy. Although a small number of responses indicated negative or unchanged values, the predominance of positive differences, supported by a highly significant p-

value, suggests a consistent upward trend in learners perceived confidence. These findings reinforce the value of embedding practical, tool-based activities in software engineering education, particularly in domains where applied security practices are critical.

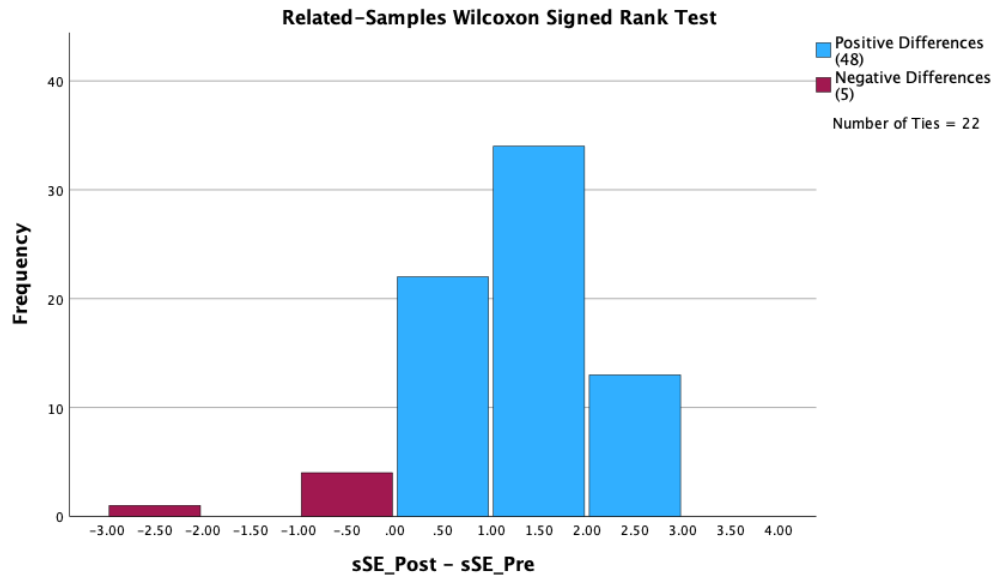


Figure 5.7. Differences between pre- and post-survey responses of sSE variable

We determined the effect size for self-efficacy to be 0.628 by applying Cohen's *d*, indicating a large effect ($d > 0.50$) [79]. Therefore, with respect to the self-efficacy variable, we can conclude that there is a statistically significant improvement in self-efficacy among participants, suggesting that the exercise successfully helped them feel more capable of using security tools in CI/CD pipelines.

Impact on Attitude

We must observe the data distribution in both the pre- and post-survey of the attitude variable before assessing the impact. Figure 5.8 represents the data distribution of the variable. We denote the attitude variables derived from the pre- and post-surveys as sAT_pre and sAT_post. The x-axis represents participants' self-assessments, which are measured on a scale from 1 to 5. A rating of 1 corresponds to "strongly disagree," while a rating of 5 indicates "strongly agree". The figure indicates that there were no significant changes in participants' attitudes toward security integration. Before the exercise, participants demonstrated a strong attitude toward security integration, and this level remained consistent throughout. Notably, the post-survey showed a slight increase in ratings, with more individuals selecting 4 or 5. This consistency in attitudes suggests that participants already valued the integration of security within the development lifecycle prior to the exercise. Such a baseline indicates a pre-existing awareness of the importance of secure development practices, which might be influenced by prior coursework, industry trends,

or individual motivation. Although the marginal increase observed in the post-survey was not substantial, the maintenance of high attitudinal levels reinforces the relevance of the topic.

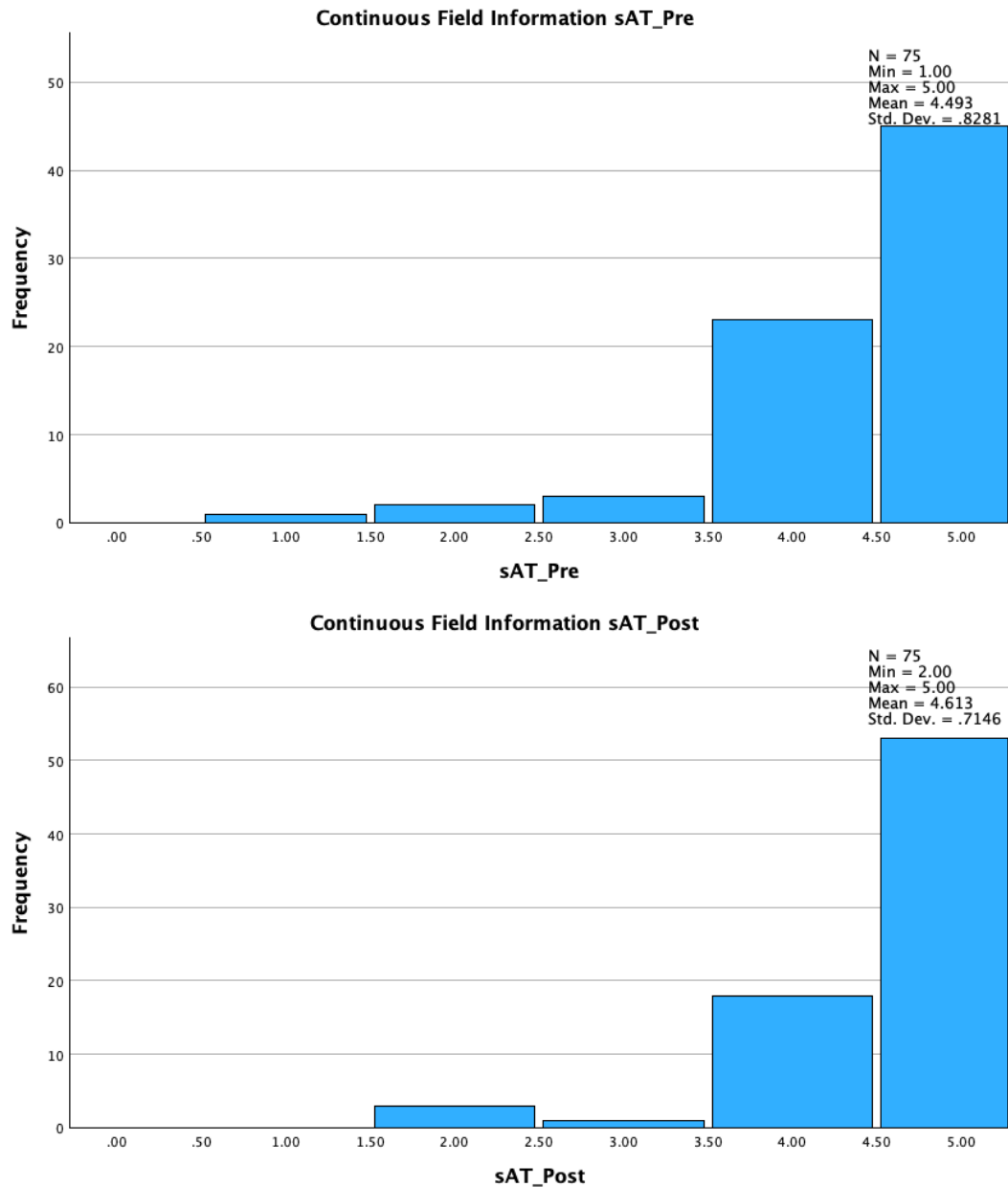


Figure 5.8. Data distribution of Attitude, sAT variable

To assess whether this improvement is statistically significant, we utilized the related-sample Wilcoxon signed-rank test. Figure 5.9 illustrates the differences between pre- and post-responses. It shows that most of the responses remained identical; however, there were slightly more positive differences, indicating that there were more instances in which the post-survey rating was higher than the pre-survey rating. Due to these small

differences, we ended up with a low Z value of -1.292. Consequently, the test could not reject the null hypothesis, as the p-value is 0.196.

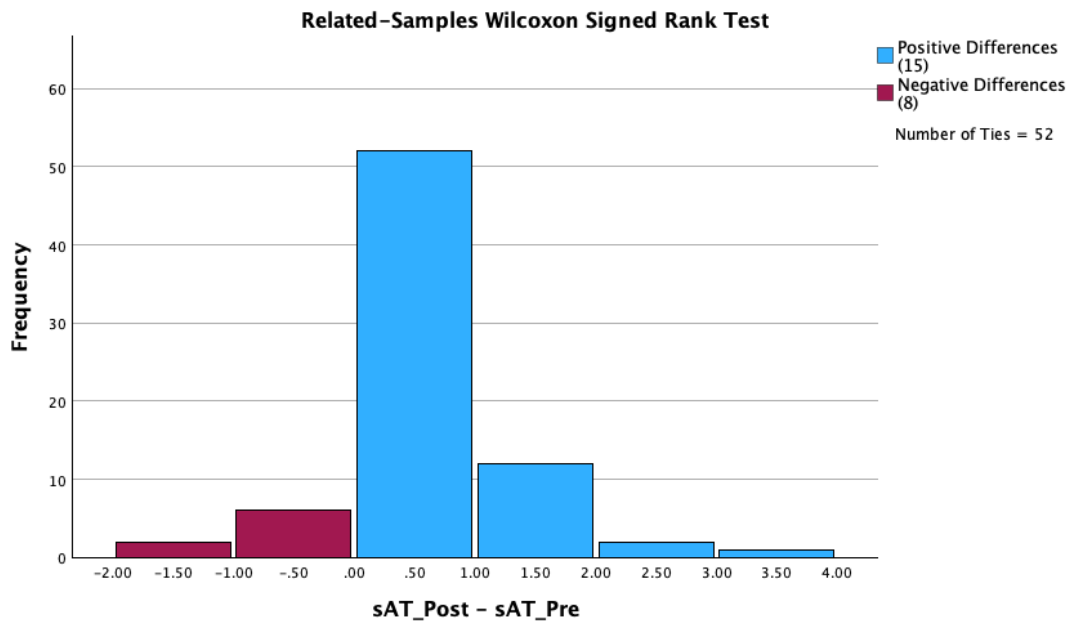


Figure 5.9. Differences between pre- and post-survey responses of sAT variable

We determined the effect size for attitude to be 0.149 by applying Cohen's d, which indicates a small effect ($d > 0.1$) [79]. Therefore, regarding this variable, we can conclude that there is no statistically significant improvement in attitude, as the participant already had a higher level of attitude toward security integration, and it remained consistent throughout the exercise.

Impact on Expectations/Outcomes

Again, before evaluating the impact on expectations and outcomes, we will first analyze the data distribution of the variable sEM in both the pre- and post-surveys. Figure 5.10 provides a visual representation of the data distribution for the variable. We refer to the expectation and outcome variables obtained from the pre- and post-surveys as sEM_pre and sEM_post. Here, sEM_pre reflects the expectations participants had regarding the exercise, while sEM_post indicates their perceived outcomes after completing it. The x-axis represents participants' self-assessments, rated on a scale from 1 to 5, where a score of 1 means "strongly disagree" and a score of 5 means "strongly agree". The results show no significant changes in participants' expectations or perceived outcomes. Before the exercise, participants had high expectations, which remained consistent in their perceived outcomes. In the post-survey, participants rated their perceived outcomes slightly higher, with an increased number of individuals giving scores of 4 or 5.

The distribution illustrated in the figure shows a notable overlap between expectations and perceived outcomes. This alignment suggests that participants had realistic anticipations of the exercise and that their actual experiences met or closely matched those expectations. The relatively high pre-survey scores indicate that learners entered the exercise with positive expectations, possibly due to prior knowledge of the topic or interest in DevSecOps. The consistency between sEM_pre and sEM_post implies that the exercise was well-aligned with participants expectations.

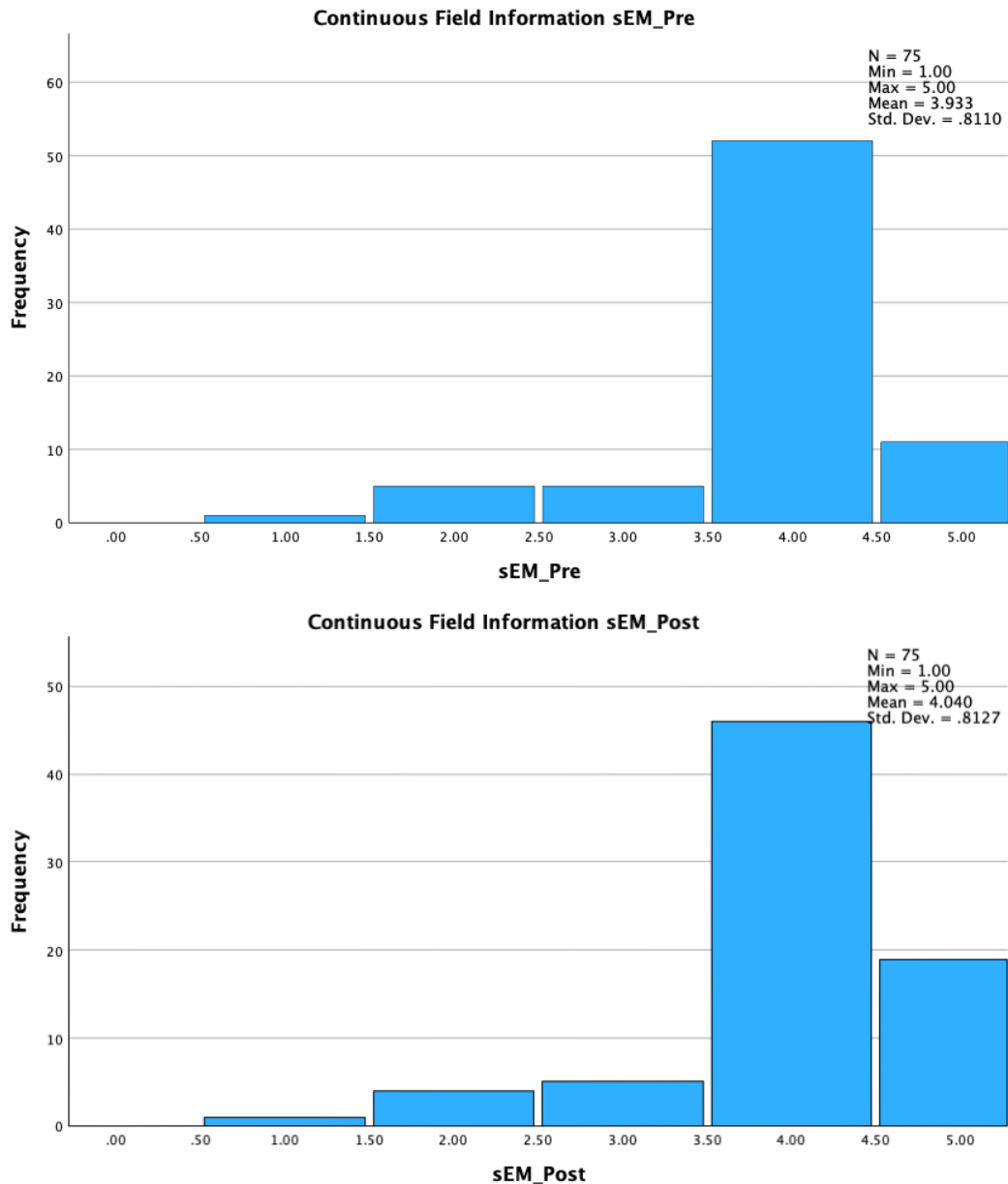


Figure 5.10. Data distribution of Expectations/Outcomes, sEM variable

To determine if these changes were statistically significant, we applied the related sample Wilcoxon signed-rank test. Figure 5.11 highlights the differences between participants' expectations and their perceived outcomes. The data reveal that most responses

were consistent; however, there were slightly more instances where the post-survey ratings exceeded the pre-survey ratings. These minor differences contributed to a low Z value of -1.003.

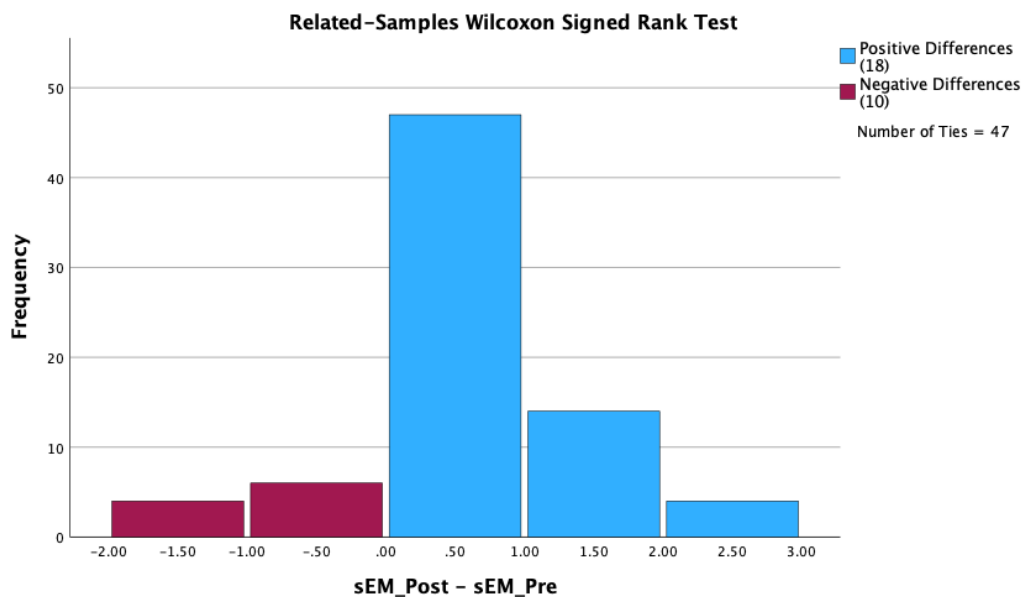


Figure 5.11. Differences between pre- and post-survey responses of sEM variable

The test was unable to reject the null hypothesis, as the p-value was 0.316. We calculated the effect size for attitude using Cohen's d, which yielded a value of 0.115, indicating a small effect ($d > 0.1$) [79]. Therefore, we conclude that there is no statistically significant difference between participants' expectations and their perceived outcomes. The result suggests that participants felt that the exercise met, and in some instances slightly exceeded their initial expectations.

Table 5.3. Results of the Hypothesis Testing and Effect Size Calculations

Variables	Z	p	Effect Size, d
Knowledge sKN	-7.077	<0.001	0.817
Interest sIN	-0.776	0.438	0.089
Self-Efficacy sSE	-5.443	<0.001	0.628
Attitude sAT	-1.292	0.196	0.149
Expectations and Outcomes sEM	-1.003	0.316	0.115

Table 5.3 presents a summary of the hypothesis testing and effect size calculations. The findings indicate that for the knowledge and self-efficacy variables, statistical significance is present, as the p-value is less than 0.05. For the other variables, since participants already exhibited higher levels in the pre-survey and maintained this consistency in the post-survey, there was no statistically significant difference between the pre- and post-

surveys. This evidence also suggests that the exercise effectively maintained this consistency for those variables. Both knowledge and interest demonstrate a substantial effect size, with d values exceeding 0.5. Conversely, the effect sizes for other variables were either very minimal or small.

5.2.4 Hypothesis Testing on Demographic Information

We categorized the participant responses based on demographic information, which includes age, gender, major, completed education level, experience in software development, and familiarity with CI/CD and DevSecOps practices. To analyze the data within these categories, we applied the Wilcoxon signed-rank test. We will now discuss the results obtained based on our variables, which include knowledge, interest, self-efficacy, attitude, and expectations or outcomes.

Analysis on Knowledge

The impact of the exercise on participants' knowledge was consistently positive across demographic groups. Table 5.4 presents the results of the hypothesis test regarding the demographic information based on the knowledge variable.

Table 5.4. Hypothesis Testing on Demographic Information for Knowledge

Demographic Information	Subcategory	Number of Responses, N	Z	p	Effect Size, d
Age	20 to 29	63	-6.408	<0.001	0.807
	30 to 39	10	-2.848	0.004	0.900
Gender	Male	55	-6.072	<0.001	0.818
	Female	18	-3.456	<0.001	0.814
Major	Information Security	32	-4.585	<0.001	0.810
	ICT	42	-5.406	<0.001	0.834
Completed Education Level	High School	12	-2.818	0.005	0.813
	Bachelor's	59	-6.336	<0.001	0.824
Experience in Software Development	No Experience	21	-3.857	<0.001	0.841
	Less than 1 Year	09	-2.514	0.012	0.838
	1 to 3 Years	34	-4.657	<0.001	0.798
	4 to 6 Years	11	-2.821	0.005	0.850
Familiarity with CI/CD	Yes	49	-5.632	<0.001	0.804
	No	26	-4.322	<0.001	0.847
Familiarity with DevSecOps	Not at all	29	-4.482	<0.001	0.832
	Slightly	38	-5.276	<0.001	0.855

Whether participants were younger (20-29) or older (30-39), both groups had statistically significant improvement as well as large effect size. Male and female participants gained nearly identical knowledge. Academic background did not appear to be a barrier for stu-

dents majoring in ICT or information security, nor for those with different levels of completed education. All participants reported significant knowledge gains. Participants with bachelor's degrees had slightly higher effect sizes than high school graduates, but both saw substantial improvements.

Experience in software development and familiarity with CI/CD or DevSecOps concepts also did not limit the effectiveness of the exercise. Even those with little to no experience gained a lot from it. This evidence suggests the exercise was accessible to beginners while still providing value to more experienced participants. We did not analyze participant subgroups with six or fewer members due to potential statistical noise. Overall, we can conclude the exercise was effective in increasing knowledge across all groups regardless of age, gender, education, or experience.

Analysis on Interest

Table 5.5 summarizes the results from the hypothesis test on demographic information related to the interest variable. The results indicate that nearly all demographic groups showed only slight changes in their interest levels, with one notable exception. This outcome is likely since participants already possessed a high level of interest in the exercise topic and it remained unchanged even after the exercise.

Table 5.5. Hypothesis Testing on Demographic Information for Interest

Demographic Information	Subcategory	Number of Responses, N	Z	p	Effect Size, d
Age	20 to 29	63	-0.329	0.727	0.041
	30 to 39	10	-0.849	0.396	0.268
Gender	Male	55	-1.018	0.309	0.137
	Female	18	-0.480	0.631	0.113
Major	Information Security	32	-0.386	0.699	0.068
	ICT	42	-1.434	0.152	0.221
Completed Education Level	High School	12	-0.241	0.809	0.069
	Bachelor's	59	-0.729	0.466	0.094
Experience in Software Development	No Experience	21	-1.329	0.184	0.290
	Less than 1 Year	09	-0.06	0.952	0.020
	1 to 3 Years	34	-2.287	0.022	0.392
	4 to 6 Years	11	-0.577	0.564	0.174
Familiarity with CI/CD	Yes	49	-0.827	0.408	0.118
	No	26	-0.244	0.807	0.047
Familiarity with DevSecOps	Not at all	29	-1.337	0.181	0.248
	Slightly	38	-0.172	0.864	0.028

Both younger (20-29) and older (30-39) participants showed little change in interest, with p-values of 0.726 and 0.396, respectively. Similarly, gender did not significantly affect participants' interest in the exercise topic. Academic background also seemed to have

little influence on the changes in interest. Likewise, education level did not play a significant role, as both high school graduates and bachelor's degree holders reported similar levels of interest. However, participants with 1 to 3 years of experience in software development showed a notable increase in interest, with a moderate effect size ($d = 0.392$, $p = 0.022$). This result suggests that those with some experience in software development found the exercise more engaging.

Conversely, familiarity with CI/CD and DevSecOps did not substantially influence interest. In summary, from the data distribution of interest in figure 5.4, we can see that participants already had a higher level of interest in the exercise topic, which remained consistent even after the exercise. Therefore, the results from table 5.5 suggest that the exercise successfully retained participants' interest, as there were no significant changes in their interest levels, even when considering their demographic information.

Analysis on Self-Efficacy

The result indicated that the exercise effectively improved participants' confidence across all demographic groups. Table 5.6 presents the results of the hypothesis test regarding the demographic information based on the self-efficacy variable.

Table 5.6. Hypothesis Testing on Demographic Information for Self-efficacy

Demographic Information	Subcategory	Number of Responses, N	Z	p	Effect Size, d
Age	20 to 29	63	-4.857	<0.001	0.612
	30 to 39	10	-2.332	0.020	0.737
Gender	Male	55	-4.518	<0.001	0.610
	Female	18	-2.950	0.003	0.695
Major	Information Security	32	-2.643	0.008	0.467
	ICT	42	-4.831	<0.001	0.745
Completed Education Level	High School	12	-2.739	0.006	0.791
	Bachelor's	59	-4.606	<0.001	0.600
Experience in Software Development	No Experience	21	-3.819	<0.001	0.834
	Less than 1 Year	09	-1.518	0.129	0.506
	1 to 3 Years	34	-3.147	0.002	0.540
	4 to 6 Years	11	-2.460	0.014	0.742
Familiarity with CI/CD	Yes	49	-4.069	<0.001	0.581
	No	26	-3.722	<0.001	0.730
Familiarity with DevSecOps	Not at all	29	-3.819	<0.001	0.709
	Slightly	38	-3.699	<0.001	0.600

Whether participants were younger (20-29) or older (30-39), both groups showed statistically significant improvements and large effect sizes. Male and female participants exhibited nearly identical gains in self-efficacy. Academic background had little impact on the results, as ICT majors demonstrated higher gains compared to those in information

security. Participants with a bachelor's degree reported slightly better self-efficacy than high school graduates, although both groups significantly benefited from the exercise.

Experience in software development also played a crucial role. Those with no experience showed notable improvements. Participants from other experience groups reported gains as well, except for the group with less than a year of experience. Finally, even individuals with minimal prior knowledge of CI/CD or DevSecOps experienced substantial growth in confidence. This evidence suggests that the exercise was accessible and valuable for both beginners and experienced participants. We did not analyze participant subgroups with six or fewer members due to potential statistical noise. Overall, we can conclude that the exercise was effective in increasing self-efficacy across almost all groups, regardless of age, gender, education, or experience.

Analysis on Attitude

Table 5.7 summarizes the results from the hypothesis test on demographic information related to the attitude variable. The results indicate that all demographic groups showed only marginal changes in their attitude levels. This outcome is likely since participants already possessed a high level of attitude toward security integration and it remained unchanged even after the exercise.

Table 5.7. Hypothesis Testing on Demographic Information for Attitude

Demographic Information	Subcategory	Number of Responses, N	Z	p	Effect Size, d
Age	20 to 29	63	-1.578	0.115	0.199
	30 to 39	10	-0.447	0.655	0.141
Gender	Male	55	-1.039	0.299	0.140
	Female	18	-1.134	0.257	0.267
Major	Information Security	32	-0.855	0.393	0.151
	ICT	42	-1.213	0.225	0.187
Completed Education Level	High School	12	0.000	1.000	0.000
	Bachelor's	59	-1.240	0.215	0.161
Experience in Software Development	No Experience	21	-0.707	0.480	0.154
	Less than 1 Year	09	0.000	1.000	0.000
	1 to 3 Years	34	-1.500	0.134	0.257
	4 to 6 Years	11	0.000	1.000	0.000
Familiarity with CI/CD	Yes	49	-0.885	0.376	0.126
	No	26	-0.973	0.331	0.191
Familiarity with DevSecOps	Not at all	29	0.000	1.000	0.000
	Slightly	38	-1.589	0.112	0.257

In terms of age, both groups exhibited no significant changes in attitude after completing the exercise. Similarly, gender did not show major differences, as both males and females experienced changes, though they were not statistically significant. For age and

gender, the effect size was small. When considering academic majors, students in both ICT and information security displayed little to no change in their attitudes. Likewise, a completed education level did not significantly impact results, as participants with high school diplomas showed no change at all ($p = 1.000$), while bachelor graduates experienced only a modest change. Participants with no experience and those with 1 to 3 years of software development experience had slight changes, but these were not statistically significant. For other groups, there were no changes whatsoever.

Finally, familiarity with CI/CD and DevSecOps had a minimal effect on attitude, with changes being small and not statistically significant. In summary, from the data distribution of attitudes in figure 5.8, it is evident that participants already held a higher level of attitude toward security integration, which remained consistent even after the exercise. Therefore, the results from table 5.7 indicate that the exercise effectively retained participants' attitudes toward security, as there were no significant changes in their attitude levels, even when considering their demographic information.

Analysis on Expectations and Outcomes

Table 5.8 shows the results from the hypothesis test on demographic information related to the expectations and outcomes variable. The results indicate that all demographic groups showed only marginal changes in participants' expectations and perceived outcomes. This result is likely since participants already had a high level of expectations toward the exercise and it remained unchanged in their assessment of outcomes as well.

Table 5.8. Hypothesis Testing on Demographic Information for Expectations and Outcomes

Demographic Information	Subcategory	Number of Responses, N	Z	p	Effect Size, d
Age	20 to 29	63	-0.895	0.371	0.112
	30 to 39	10	-1.000	0.317	0.316
Gender	Male	55	-1.119	0.263	0.151
	Female	18	-0.107	0.915	0.025
Major	Information Security	32	0.000	1.000	0.000
	ICT	42	-1.252	0.210	0.193
Completed Education Level	High School	12	-1.190	0.234	0.343
	Bachelor's	59	-0.495	0.621	0.065
Experience in Software Development	No Experience	21	-1.265	0.206	0.276
	Less than 1 Year	09	-0.378	0.705	0.126
	1 to 3 Years	34	-1.230	0.222	0.210
	4 to 6 Years	11	-0.414	0.679	0.125
Familiarity with CI/CD	Yes	49	-0.884	0.377	0.126
	No	26	-0.486	0.627	0.095
Familiarity with DevSecOps	Not at all	29	-0.253	0.800	0.047
	Slightly	38	-0.924	0.356	0.150

In terms of age, both younger and older groups showed little to no changes in their expectations and outcomes, with small effect sizes of 0.112 and 0.316. Similarly, gender did not significantly affect the results, as both male and female participants experienced minimal changes. When considering academic majors, students from information security showed no changes at all, while ICT students exhibited minor changes that were not statistically significant. Likewise, education level also did not significantly influence the outcomes, as both groups experienced minor changes. Participants with no experience and those with 1 to 3 years of experience achieved better results compared to other groups, but these were not statistically significant either.

Finally, familiarity with CI/CD and DevSecOps had minimal effects on the variable. While those familiar with CI/CD showed a small change, and those with slight familiarity with DevSecOps experienced a slightly larger change, neither was statistically significant. In summary, based on the data distribution of expectations and outcomes in Figure 5.10, it is evident that participants held higher expectations, which remained consistent in their assessment of perceived outcomes. Therefore, the results from Table 5.8 indicate that the exercise effectively met participants' expectations, as there were no significant changes in their perceived outcomes, even when considering their demographic information.

5.3 Qualitative Analysis

This section presents the results of the qualitative analysis of responses to open-ended feedback questions in the post-survey. The goal was to gain a more profound understanding of how the participants experienced the exercise. This information includes aspects of the exercise they found helpful, scopes of improvement, and their key takeaways from it. A total of 75 responses were analyzed using thematic coding in MAXQDA24.

5.3.1 Helpful Aspects of the Exercise

Participants identified multiple sections of the exercise which they considered particularly helpful for enhancing their learning experience. Table 5.9 presents the generated codes and themes derived from their responses to this question. Each code reflects the section that participants found helpful, which was subsequently organized into themes. In this instance, most codes remained consistent within their themes, apart from exercise instructions and video tutorials, which were merged into the "Exercise Materials" theme. Thus, through thematic analysis of the responses, six unique themes were identified.

Table 5.9. *Generated codes and themes for the Helpful Aspects of the Exercise*

Codes	Frequency	Themes
Hands-on Experience	04	Hands-on Experience
Exercise Session	02	Exercise Session
Exercise Instructions	10	Exercise Materials
Video Tutorials	04	
Analyzing Artifacts for Vulnerability	03	Analyzing Artifacts for Vulnerability
Implementation of Secure CI/CD Pipeline	36	Implementation of Secure CI/CD Pipeline
Learning Different Tools	25	Learning Different Tools

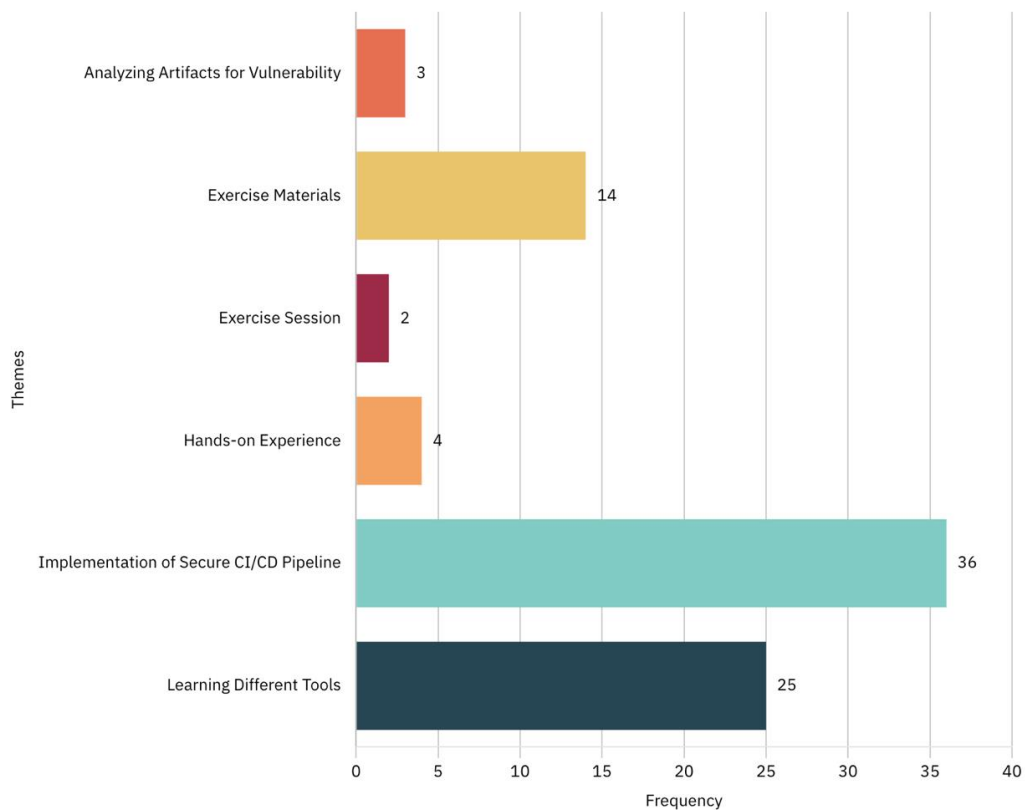
**Figure 5.12.** *Frequency of themes for the Helpful Aspects of the Exercise*

Figure 5.12 represents the frequency of the identified themes for this question. The most frequently mentioned section was the “Implementation of a secure CI/CD pipeline,” as it enabled participants to interact with security concepts in a real-world environment. Exposure to different security tools was emphasized as highly beneficial. This was because it equipped the participants with hands-on understanding of industry-relevant technologies. Participants acknowledged that exercise materials were highly beneficial in guiding them through the task. Some participants appreciated the hands-on nature of the exercise, as it was able to clarify their understanding. Additionally, participants valued the opportunity to analyze the artifacts for vulnerabilities as well as the exercise session.

5.3.2 Scope of Improvement

When it comes to improvements, participants offered constructive feedback meant to improve the exercise experience. Table 5.10 presents the generated codes and themes derived from their responses to this question. Each code reflects suggestions from participants, which was subsequently organized into themes. Thus, through thematic analysis of the responses, five unique themes were identified.

Table 5.10. Generated codes and themes for the Scopes of Improvement

Codes	Frequency	Themes
More exercise sessions	02	Improve Time Management and Structure
Execute in multiple weeks	26	
Real-world test application	01	Make Exercise more Realistic
Reduce the exercise tasks	03	Simplify and Balance the Exercise Tasks
Reduce the pipeline execution time	07	
Addition of more complex tasks	03	Increase complexity in the Exercise
Troubleshooting guide	06	Enhance Guidance and Support
Exercise instructions	12	

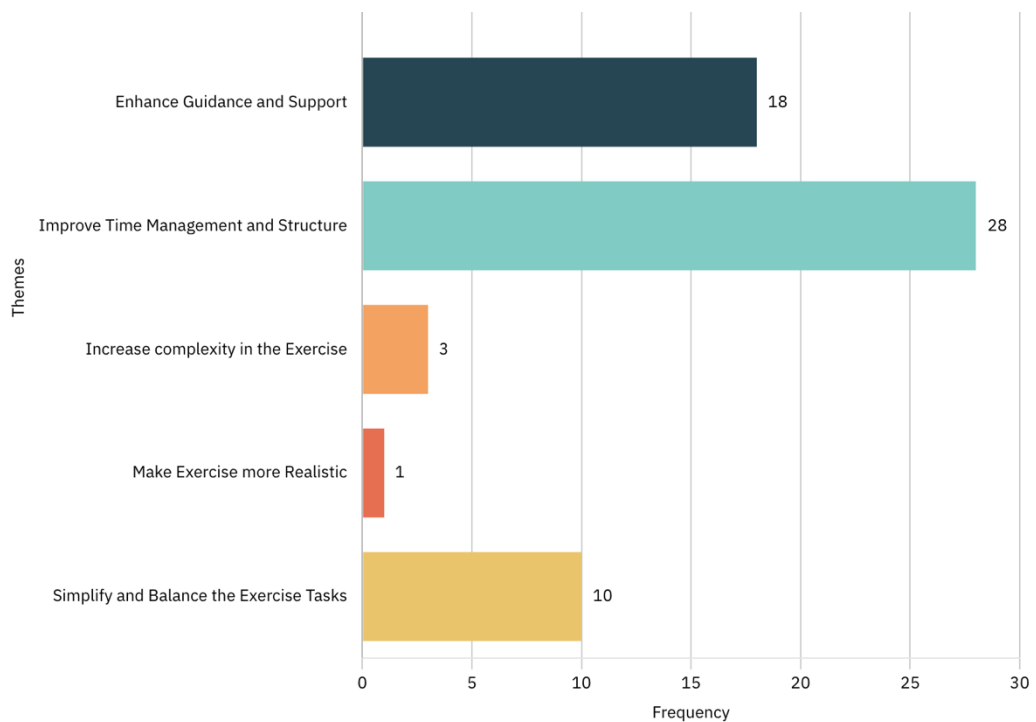


Figure 5.13. Frequency of themes for the Scopes of Improvement

Figure 5.13 represents the frequency of the identified themes for this question. An important suggestion from participants was the need for better time management. This clearly indicates that time-related issues were the primary concern among participants. They wanted the exercise to be distributed over a longer period of the time to allow for

deeper learning and reduced cognitive load. Participants also expressed the need more clearer instructions as well as troubleshooting supports. While some respondents recommended simplifying certain tasks, others proposed increasing the complexity level to better challenge their abilities. There was also an interest in incorporating more real-world test application to make the exercise feel more authentic.

5.3.3 Key Takeaways from the Exercise

Reflecting on the exercise, participants shared several key learning outcomes. Table 5.11 presents the generated codes and themes derived from their responses to this question. Each code reflects key takeaways from participants, which was subsequently organized into themes. In this instance, all codes remained consistent within their themes. Thus, through thematic analysis of the responses, four unique themes were identified.

Table 5.11. Generated codes and themes for the Key Takeaways from the Exercise

Codes	Frequency	Themes
Significance in Security and DevSecOps	16	Significance in Security and DevSecOps
Analyzing artifacts for Vulnerability	08	Analyzing artifacts for Vulnerability
Experience on Different Security Testing Tools	17	Experience on Different Security Testing Tools
Understanding and Setting up a security-focused CI/CD Pipeline	41	Understanding and Setting up a security-focused CI/CD Pipeline

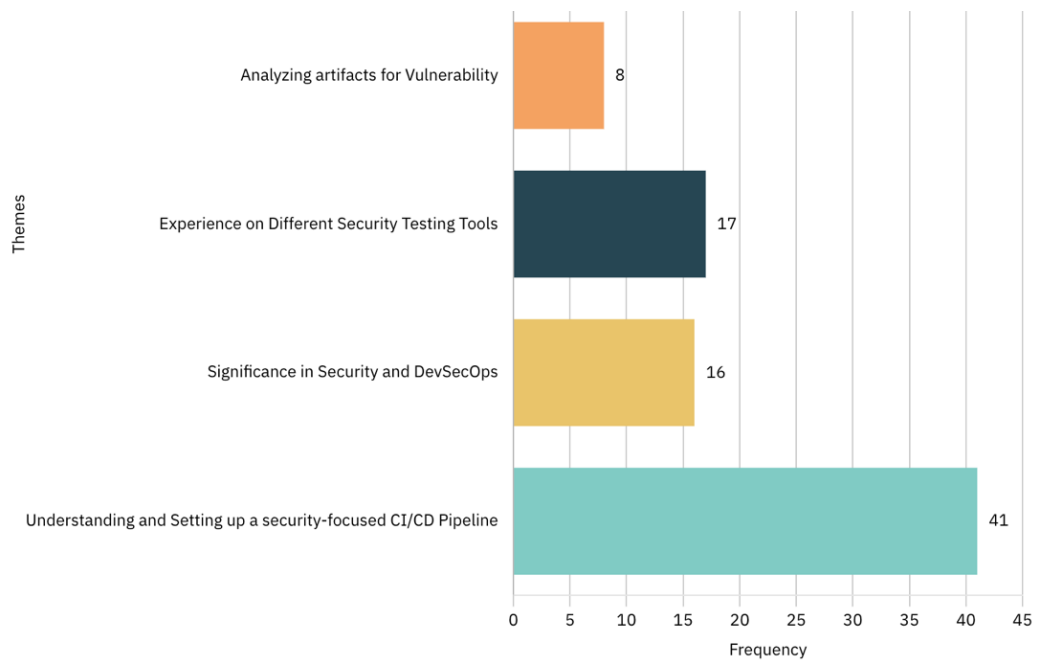


Figure 5.14. Frequency of themes for the Key Takeaways from the Exercise

Figure 5.14 represents the frequency of the identified themes for this question. As a critical takeaway from this exercise, many participants emphasized a stronger understanding of how to design and operate a secure CI/CD pipeline. Participants also highlighted the opportunity to work directly with multiple security testing tools. Participants felt increased confidence in applying these tools in real-world environments. Many also appreciated the role of security in software development, particularly embedding it early in the development life cycle. Another important reported outcome was the participants' ability to analyze artifacts for vulnerabilities. Together, these key learning points indicate that the exercise was able to successfully connect theoretical concepts with practical skills.

Overall, the qualitative analysis presented in this section highlights the effectiveness of the exercise in developing participants' knowledge, skills, and confidence in DevSecOps practices. Participants provided insights into both strengths and areas of improvement for this exercise. They underscored the importance of hands-on experience with security-focused pipelines and security testing tools. Their constructive feedback, like improving the time management and clear instructions, highlighted the opportunity for refining the structure and execution in future implementations. These insights confirm the importance of this exercise and offer a roadmap for improvement.

6. DISCUSSION

This chapter discusses the key findings of the study and places them in the context of existing literature. It begins by addressing the research questions, followed by a comparison with related work in the fields of DevSecOps and software security education. The goal is to highlight the contribution of the developed work and reflect on its implications for future educational practices.

6.1 Addressing the Research Questions

The first research question, RQ1 explored how to design an exercise that incorporates DevSecOps concepts into software development education. The developed exercise integrated essential security practices and tools, including Static Application Security Testing (SAST), Software Composition Analysis (SCA), Dynamic Application Security Testing (DAST), container image scanning, and file system security scans. Additionally, the exercise adopted key DevSecOps principles, such as unifying the CI/CD pipeline, enabling fail-fast automation, defining relevant security checkpoints, and securing development environments and toolchains. It utilized Jenkins as the automation server for the CI/CD pipeline, along with security tools like SonarQube, Snyk, OWASP ZAP, and Trivy. This approach aligns with the "shift-left" philosophy of DevSecOps, which emphasizes integrating security early and continuously throughout the software development lifecycle. In this way, the developed exercise was able to incorporate DevSecOps concepts.

The second research question, RQ2 examined how the exercise impacted participants' knowledge, interest, self-efficacy, and attitudes toward secure software development. The quantitative analysis presented in the previous chapter showed significant improvements in participants' knowledge and self-efficacy. They reported a substantial increase in their understanding and ability to use various security tools, indicating the effectiveness of the developed exercise. However, the participants' interest and attitudes toward secure software development did not change significantly. This lack of change can be attributed to the ceiling effect, which indicates that they already possessed a high level of interest and a positive attitude toward security integration. This consistency before and after the exercise highlights the participants' recognition of the significance of DevSecOps and the exercise.

The third research question, RQ3 examined the degree to which participants' expectations for the exercise aligned with their perceived outcomes. Both quantitative and qualitative analyses indicated that participants' expectations closely matched their perceived results. In the quantitative analysis, participants exhibited high expectations for the exercise, which remained consistent after its completion. Although statistical significance was not reached due to already high baseline expectations, qualitative analysis revealed that most participants felt the exercise met or exceeded their expectations. This alignment highlights the effectiveness of the developed exercise in providing a realistic and beneficial educational experience for the audience.

The fourth research question, RQ4, asked to what degree demographic information such as age and gender influences participants' knowledge, interest, self-efficacy, attitude, and perceived outcomes. Overall, the demographic factors did not significantly affect the exercise's impact. This finding indicates that the exercise was accessible to all participants regardless of factors such as age, gender, education, or experience. However, there are certain exceptional cases. For example, in terms of the interest variable, even though there were not significant changes in most demographic groups, participants with 1 to 3 years of experience in software development showed a notable increase in interest, with a moderate effect size. This result suggests that those with some experience in software development found the exercise more engaging. In terms of self-efficacy, most groups reported significant changes except for the group with less than a year of experience. However, the lower sample size of this group suggests the result might be some statistical noise. For other variables, which are knowledge, attitude, and expectations/outcomes, there were no exceptional cases. The evidence suggests demographic information did not have any significant effect on all variables except for the interest variable, where participants with 1–3 years of experience in software development saw some significant changes in their interest.

6.2 Comparison to Related Work

In reviewing previous literature, it became clear that DevSecOps education is still largely an unexplored area. In fact, the literature search identified only one article, by M. M. Rahman et al. [71], that directly examined how DevSecOps principles can be integrated into educational environment. They primarily described a practical exercise involving Git Hooks and static analysis tools, aiming to increase students' awareness of security issues within DevOps. However, their research was limited in its evaluation approach, fo-

cusing mostly on specific technical skills rather than assessing broader educational impacts. In terms of assessment, they had around 20 pre-responses and 09 post-responses.

Other reviewed articles typically fell into two separate categories: general DevSecOps implementations [53–60] and broader approaches to software security education [61–70]. Although these studies provide important insights into security tools, vulnerability detection strategies, and general teaching methodologies, none specifically combined the educational aspects of DevSecOps into a unified exercise. This thesis expanded significantly upon these earlier efforts. Unlike M.M. Rahman et al. [71], it introduced a complete, realistic DevSecOps pipeline exercise that incorporated multiple widely used security tools such as SonarQube, Snyk, OWASP ZAP, and Trivy. By exposing participants to a holistic pipeline that closely resembles real-world development processes, the exercise facilitated a more authentic and practical learning experience compared to prior isolated approaches.

Furthermore, this research adopted a broader and more detailed evaluation approach. The data analysis included a dataset containing 75 complete pre- and post-responses. Rather than limiting the assessment to technical measures alone, this study systematically examined multiple variables—such as participants' knowledge, interest, self-efficacy, attitudes toward security, and alignment of expectations with outcomes. This approach provided a more nuanced understanding of how effectively students absorbed and engaged with the content. Additionally, participant feedback offered valuable qualitative insights into areas that could improve the learning experience, such as clearer instructions and more balanced workload distribution. This type of feedback, rarely addressed in prior work, provides practical guidance for improving future DevSecOps exercises.

Moreover, while existing literature discussed earlier provided valuable insights into technical and awareness-building aspects of DevSecOps education, it has generally overlooked comprehensive evaluations involving broader cognitive and motivational outcomes (Research Question 2-4). Previous studies, such as those by Rahman et al. [71] and Kumar and Goyal [53], primarily emphasized technical skills development and specific security tools, without systematically evaluating the psychological and motivational dimensions supporting learning effectiveness. In contrast, this thesis explicitly evaluated variables such as knowledge, interest, self-efficacy, attitudes, and the alignment of expectations with perceived outcomes.

In line with earlier findings by Schafeitel-Tähtinen et al. [79] and Bell et al. [86], participants in this study demonstrated significant improvements in both knowledge and self-efficacy. Specifically, these outcomes confirm that structured and practical cybersecurity exercises effectively enhance students' understanding of security tools and their confidence in performing secure development tasks. Such results reinforce the validity of these variables in assessing educational impact, highlighting the strength of hands-on DevSecOps exercises in producing measurable gains in learner competence.

However, the minimal observed changes in interest and attitudes contrast somewhat with theoretical predictions derived from Hidi's [90] conceptualization of situational interest and Faklaris et al.'s [88] insights into attitudinal change. This discrepancy likely arises from a ceiling effect, as the participants' initially high interest levels and already positive attitudes toward security integration left limited room for measurable improvement within the short timeframe of the exercise. This finding supports the perspective presented by Flowerday and Shell [89], who suggested that substantial motivational and attitudinal changes typically require longer-term or personalized educational engagements rather than short-term interventions.

Demographic analyses in this thesis provided additional, nuanced understanding compared to previous literature. Studies by Colin et al. [95] and Mumtaz et al. [51] indicated experience as a moderating factor influencing cybersecurity education outcomes. Consistent with these earlier studies, this research identified a notable increase in interest specifically among participants with moderate professional experience (1–3 years). This suggests prior familiarity with relevant software development contexts may enhance engagement with DevSecOps, thus further clarifying how educational effectiveness can vary depending on learner backgrounds.

Regarding the assessment instrument itself, the employed measurement tool demonstrated strong internal consistency and effectively captured improvements in knowledge and self-efficacy, as indicated by high reliability scores. However, the instrument exhibited limited sensitivity in detecting subtle shifts in interest and attitudes, likely due to participants' high initial scores on these dimensions. This limitation highlights the challenge of evaluating incremental motivational or attitudinal changes among already highly motivated learners. Consequently, the results validate the effectiveness of the measurement approach for clearly observable cognitive variables but suggest opportunities for improved sensitivity in measuring motivational outcomes.

Overall, by integrating a realistic pipeline exercise with a comprehensive, learner-focused evaluation framework, this thesis not only addressed a clear research gap but also

provided meaningful recommendations for future practices in secure software development education.

7. CONCLUSIONS

The main purpose of this study was to explore how DevSecOps can be integrated into software engineering education using a hands-on exercise. This study developed and evaluated a secure CI/CD pipeline made with widely used tools like SonarQube, Snyk, OWASP ZAP, and Trivy to give students experience that is as close as possible to actual development environments. The purpose of the exercise was to develop both technical skills and a mindset that values early and continuously integrated security. In this analysis, there were few clear outcomes. First, the participants exhibited significant changes in their knowledge and self-efficacy. This evidence would suggest that the exercise helped participants comprehend key security practices and feel more capable of performing them. Second, while there was no change in either interest or attitudes, it seems likely that these factors were already high prior to the exercise. Therefore, we can interpret their results as positive. The findings indicate that the exercise closely matched the participants' expectations and sustained their motivation during it. Another significant aspect of the findings is that the exercise benefits extended across demographic categories. Improvement was apparent across a wide variety of participants regardless of age, educational level, and prior experience. This strengthens the argument of the exercise being a widely available and accessible learning tool. Collected feedback also suggested important changes for improving future implementations, such as simplifying instructions, offering more guidance, and adjusting the workload. Through the review of the current literature, it was clear that while many studies investigated DevSecOps tools and practices, there was a lack of research on how these practices could be taught. There was only one identified work that specifically focused on DevSecOps in an educational context, and it did not include the same level of evaluation that is undertaken in the current study. Through the combination of a technical framework with an evaluation of an educational framework, this thesis offers a more thorough approach to teaching secure software development. Overall, this study provides a useful template for integrating security into software education. The evidence shows that exercises that have been designed appropriately can enhance understanding and self-efficacy of the participants. As the industry increasingly demands these skills, such educational efforts will become more relevant and necessary.

7.1 Limitations and Future Work

While this thesis presents valuable findings, certain limitations should be considered. The exercise was conducted in a single academic environment, within the scope of one course. This situation naturally limits the generalizability of the results to other institutions or educational contexts. However, the consistency of the outcomes among participants with different backgrounds suggests potential applicability beyond this study. To better understand the broader relevance, future studies could implement similar exercises in different institutional or cultural contexts.

Another limitation concerns the method of evaluation. The study relied primarily on self-reported measures to assess the variables. While these offer important insights into perceived learning, they do not necessarily reflect actual skill acquisition. Introducing practical, performance-based assessments could provide a more accurate picture of students' capabilities. Additionally, limited changes observed in participants' motivation and attitudes may be attributed to a ceiling effect. Many participants began the exercise with high levels of interest and a positive disposition toward security, which left little room for measurable improvement. This highlights a broader challenge in educational research which is to evaluate incremental motivational or attitudinal shifts among already highly motivated learners. Future studies may consider using more sensitive instruments or longer-term evaluations to better capture these subtle changes over time.

Moreover, the survey questions used for evaluation were not based on a formally established structural framework for DevSecOps. While their design was adopted from the work of Schafeitel-Tähtinen et al. [79], it is important to note that their framework was originally developed for evaluating cybersecurity education, not specifically DevSecOps. As a result, the applicability of their structure may be limited in this context. Future research should aim to develop a validated, domain-specific evaluation framework tailored to secure software development and DevSecOps education. Such a framework would enhance the reliability and generalizability of the assessment.

Looking ahead, it would be beneficial to investigate how the exercise performs when integrated into multiple courses or extended over longer periods. This would enable exploration of the long-term effects of such interventions on both technical understanding and security mindset. Furthermore, exploring alternative toolchains may assist in customizing the exercise for various instructional requirements or technical preferences. Lastly, the study did not address institutional or instructor-level challenges that may affect

the adoption of DevSecOps education such as curriculum constraints, instructor familiarity with security tools, or access to infrastructure. These are important considerations for scaling the approach more broadly. These developments would contribute to a more adaptable and comprehensive approach to DevSecOps education.

REFERENCES

- [1] Härkönen, H., Lakoma, S., Verho, A., Torkki, P., Leskelä, R., Pennanen, P., Laukka, E., & Jansson, M. (2024). Impact of digital services on healthcare and social welfare: An umbrella review. *International Journal of Nursing Studies*, 152, 104692. <https://doi.org/10.1016/j.ijnurstu.2024.104692>
- [2] Dupont, B., Shearing, C., Bernier, M., & Leukfeldt, R. (2023). The tensions of cyber-resilience: From sensemaking to practice. *Computers & Security*, 132, 103372. <https://doi.org/10.1016/j.cose.2023.103372>
- [3] Martínez, J., & Durán, J. M. (2021). Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case study. *International Journal of Safety and Security Engineering*, 11(5), 537–545. <https://doi.org/10.18280/ijisse.110505>
- [4] Adesola, H., Chen, L., Ji, Y., & Kim, J. (2025). Application of Robotics Process Automation to the MOVEIT attack: A case study. In *Communications in computer and information science* (pp. 503–515). https://doi.org/10.1007/978-3-031-86637-1_37
- [5] Nearly half of US citizens hit by massive Equifax breach. (2017). *Computer Fraud & Security*, 2017(9), 1–3. [https://doi.org/10.1016/s1361-3723\(17\)30094-5](https://doi.org/10.1016/s1361-3723(17)30094-5)
- [6] Wortman, P. A., & Chandy, J. A. (2022). A framework for evaluating security risk in system design. *Discover Internet of Things*, 2(1). <https://doi.org/10.1007/s43926-022-00027-w>
- [7] Prates, L., & Pereira, R. (2024). DevSecOps practices and tools. *International Journal of Information Security*, 24(1). <https://doi.org/10.1007/s10207-024-00914-z>
- [8] Zhao, X., Clear, T., & Lal, R. (2024). Identifying the primary dimensions of DevSecOps: A multi-vocal literature review. *Journal of Systems and Software*, 214, 112063. <https://doi.org/10.1016/j.jss.2024.112063>
- [9] Mothanna, Y., ElMedany, W., Hammad, M., Ksantini, R., & Sharif, M. S. (2024). Adopting security practices in software development process: Security testing framework for sustainable smart cities. *Computers & Security*, 144, 103985. <https://doi.org/10.1016/j.cose.2024.103985>
- [10] Cope, R. (2020). Strong security starts with software development. *Network Security*, 2020(7), 6–9. [https://doi.org/10.1016/s1353-4858\(20\)30078-7](https://doi.org/10.1016/s1353-4858(20)30078-7)
- [11] Goupil, F., Laskov, P., Pekaric, I., Felderer, M., Dürr, A., & Thiesse, F. (2022). Towards understanding the skill gap in cybersecurity. *arXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.2204.13793>

- [12] Towhidi, G., & Pridmore, J. (2023). Aligning Cybersecurity in Higher Education with Industry Needs. *Journal of Information Systems Education*, 34(1), 70–83.
- [13] Langer, A. M. (2012). *Guide to Software Development: Designing and Managing the Life Cycle*. In Springer eBooks. <https://doi.org/10.1007/978-1-4471-2300-2>
- [14] Leach, R. J. (2018). *Introduction to software engineering*. In Chapman and Hall/CRC eBooks. <https://doi.org/10.1201/9781315371665>
- [15] Pressman, R. S. (1994). *Software Engineering: A Practitioner's Approach*. McGraw-Hill.
- [16] Sommerville, Ian. (2016). *Software Engineering* (Ed. 10th). Boston: Pearson
- [17] The seven phases of the software development life cycle. (2025, May 4). Harness.io. <https://www.harness.io/blog/software-development-life-cycle-phases>
- [18] Saravanos, A., & Curinga, M. X. (2023). Simulating the software development lifecycle: the Waterfall model. *Applied System Innovation*, 6(6), 108. <https://doi.org/10.3390/asi6060108>
- [19] Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Pearson Education.
- [20] Shylesh, S. (2017). A study of software development life cycle process models. *SSRN Electronic Journal*. <https://doi.org/10.2139/ssrn.2988291>
- [21] Cui, J. (2024). The Role of DevOps in Enhancing Enterprise Software Delivery Success through R&D Efficiency and Source Code Management. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2411.02209>
- [22] Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and its practices. *IEEE Software*, 33(3), 32–34. <https://doi.org/10.1109/ms.2016.81>
- [23] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5, 3909–3943. <https://doi.org/10.1109/access.2017.2685629>
- [24] Meyer, M. (2014). Continuous integration and its tools. *IEEE Software*, 31(3), 14–16. <https://doi.org/10.1109/ms.2014.58>
- [25] Marijan, D., Liaaen, M., & Sen, S. (2018). DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration. 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 1, 22–27. <https://doi.org/10.1109/COMPSAC.2018.00012>
- [26] Alanda, A., Mooduto, H. A., & Hadelina, R. (2022). Continuous integration and Continuous Deployment (CI/CD) for web applications on cloud infrastructures.

- JITCE (Journal of Information Technology and Computer Engineering), 6(02), 50–56. <https://doi.org/10.25077/jitce.6.02.50-56.2022>
- [27] Laster, B. (2020). Continuous Integration vs. Continuous Delivery vs. Continuous Deployment, 2nd Edition (Segunda edición). O'Reilly Media, Inc.
- [28] Cyberfella. (2020, March 23). What is DevOps? @Cyberfellabtc, Since 2012. Retrieved May 11, 2025, from <https://www.cyberfella.co.uk/2020/03/23/what-is-devops/>
- [29] Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2103.08266>
- [30] Gupta, A. (2022). An Integrated Framework for DevSecOps Adoption. arXiv.Org. <https://doi.org/10.48550/arxiv.2207.04093>
- [31] Expert, G. (2025, January 30). DevSecOps Introduction for beginners: Security in the SDLC - GitGuardian Blog. GitGuardian Blog - Take Control of Your Secrets Security. <https://blog.gitguardian.com/devsecops-introduction-accelerating-software-development/>
- [32] Abed Saif Ahmed Alghawli, & Radivilova, T. (2024). Resilient Cloud cluster with DevSecOps security model, automates a data analysis, vulnerability search and risk calculation. arXiv.Org. <https://doi.org/10.48550/arxiv.2412.16190>
- [33] Czekster, R. M. (2024). Continuous risk assessment in secure DevOps. arXiv.Org.
- [34] Sehgal, V. V. (2023). Implementing DevSecOps Practices: Supercharge Your Software Security with DevSecOps Excellence. (First edition). Packt Publishing Ltd.
- [35] Turner, R. (2021). Systems Engineering and DevSecOps: Reviewing the principles. Insight, 24(2), 38–43. <https://doi.org/10.1002/inst.12339>
- [36] Vandana Verma Sehgal. (2023). Implementing DevSecOps Practices. Packt Publishing.
- [37] Prates, L., & Pereira, R. (2024). DevSecOps practices and tools. International Journal of Information Security, 24(1). <https://doi.org/10.1007/s10207-024-00914-z>
- [38] Sehgal, V. V. (2023). Implementing DevSecOps Practices: Understand Application Security Testing and Secure Coding by Integrating SAST and DAST (1st ed.). Packt Publishing, Limited.

- [39] Ivanova, E., Stakhanova, N., & Sistany, B. (2024). Adversarial analysis of software composition analysis tools. In *Lecture notes in computer science* (pp. 161–182). https://doi.org/10.1007/978-3-031-75764-8_9
- [40] Verdet, A., Hamdaqa, M., Leuson Da Silva, & Khomh, F. (2023). Exploring Security Practices in Infrastructure as Code: An Empirical Study. *arXiv.Org*.
- [41] Rice, L. (2020). *Container security: fundamental technology concepts that protect containerized applications* (1st edition). O'Reilly Media.
- [42] Sultan, S., Ahmad, I., & Dimitriou, T. (2019). Container security: issues, challenges, and the road ahead. *IEEE Access*, 7, 52976–52996. <https://doi.org/10.1109/access.2019.2911732>
- [43] Devi Priya, V. S., Sethuraman, S. C., & Khan, M. K. (2023). Container security: Precaution levels, mitigation strategies, and research perspectives. *Computers & Security*, 135, 103490. <https://doi.org/10.1016/j.cose.2023.103490>
- [44] Green Jeremy. (2024). Security Lifecycle and DevSecOps. In *Information Security Management Principles* (4th Edition, pp. 1–2). BCS The Chartered Institute for IT.
- [45] Hsu, T. H.-C. (2018). *Hands-On Security in DevOps: Ensure Continuous Security, Deployment, and Delivery with DevSecOps* (1st edition.). Packt Publishing, Limited.
- [46] Net Solutions. (2025, May 5). What is DevSecOps: Definition, Challenges, and Best Practices. Retrieved May 12, 2025, from <https://www.netsolutions.com/insights/what-is-devsecops/>
- [47] OWASP Top Ten | OWASP Foundation. (n.d.). Retrieved May 12, 2025, from <https://owasp.org/www-project-top-ten/>
- [48] GitLab 2024 Global DevSecOps Report | GitLab. (n.d.). *about.gitlab.com*. Retrieved April 10, 2025, from <https://about.gitlab.com/developer-survey/>
- [49] OWASP University Challenge. (n.d.). Retrieved March 15, 2025, from <https://2017.appsec.eu/program/university-challenge>
- [50] Force, C. T. (2020). *Computing Curricula 2020: Paradigms for Global Computing Education*. New York, NY, USA: Association for Computing Machinery.
- [51] Mumtaz Abdul Hameed, & Nalin Asanka Gamagedara Arachchilage. (2018). Understanding the influence of Individual's Self-efficacy for Information Systems Security Innovation Adoption: A Systematic Literature Review. *arXiv.Org*.
- [52] Larios-Vargas, E., Elazhary, O., Yousefi, S., Lowlind, D., Vliek, M. L. W., & Storey, M. (2023). DASP: a framework for driving the adoption of software security

- practices. *IEEE Transactions on Software Engineering*, 49(4), 2892–2919. <https://doi.org/10.1109/tse.2023.3235684>
- [53] Kumar, R., & Goyal, R. (2020). Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC). *Computers & Security*, 97, 101967. <https://doi.org/10.1016/j.cose.2020.101967>
- [54] Marandi, M., Bertia, A., & Silas, S. (2023). Implementing and Automating Security Scanning to a DevSecOps CI/CD Pipeline. *2023 World Conference on Communication & Computing (WCONF)*, 1–6. doi:10.1109/WCONF58270.2023.10235015
- [55] Sedrakyan, G., Iacob, M.-E., & Van Hillegersberg, J. (2024). Towards LowDevSecOps Framework for Low-Code Development: Integrating Process-Oriented Recommendations for Security Risk Management. *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 886–894. Presented at the Linz, Austria. doi:10.1145/3652620.3688335
- [56] Saurabh, S. K., & Kumar, D. (2024). Model to reduce DevOps pipeline execution time using SAST. *International Journal of Systems Assurance Engineering and Management*, 15(5), 1999–2009. <https://doi.org/10.1007/s13198-024-02262-6>
- [57] Drane, L., McDonnell, M., Petras, R., Stiner, C., Ruckman, A. J., Wiggins, G. M., Cage, G., Smith, R., Hitefield, S., McGaha, J., Ayres, A., Brim, M., Archibald, R., & Malviya-Thakur, A. (2024). Integrating scientific single-page applications with DevSecOps. *Future Generation Computer Systems*, 166, 107695. <https://doi.org/10.1016/j.future.2024.107695>
- [58] Ramaj, X., Sánchez-Gordón, M., Gkioulos, V., & Colomo-Palacios, R. (2024). On DevSecOps and Risk Management in Critical Infrastructures: Practitioners' Insights on Needs and Goals. *Proceedings of the 2024 ACM/IEEE 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability*, 45–52. Presented at the Lisbon, Portugal. doi:10.1145/3643662.3643954
- [59] Casola, V., De Benedictis, A., Mazzocca, C., & Orbinato, V. (2023). Secure software development and testing: A model-based methodology. *Computers & Security*, 137, 103639. <https://doi.org/10.1016/j.cose.2023.103639>
- [60] Singh, R., Yeboah-Ofori, A., Kumar, S., & Ganiyu, A. (2024). Fortifying Cloud DevSecOps Security Using Terraform Infrastructure as Code Analysis Tools. *2024 International Conference on Electrical and Computer Engineering Researches (ICECER)*, 1–6. doi:10.1109/ICECER62944.2024.10920371
- [61] Liu, Y., & Ezenwoye, O. (2023). Use of SecureED as a tool for software Security education: An experience report. *2021 IEEE Frontiers in Education Conference (FIE)*, 1–7. <https://doi.org/10.1109/fie58773.2023.10343394>

- [62] Shahriar, H., Qian, K., Shalan, A., & Wu, F. (2020). Enhancing Proactive Control Mobile and Web Software Security Education with Hands-on Labware. 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 1095–1096. <https://doi.org/10.1109/compsac48688.2020.0-123>
- [63] Xie, T., Bishop, J., Tillmann, N., & de Halleux, J. (2015). Gamifying software security education and training via secure coding duels in code hunt. Proceedings of the 2015 Symposium and Bootcamp on the Science of Security. Presented at the Urbana, Illinois. doi:10.1145/2746194.2746220
- [64] Zouahi, H. (2024). Gamifying Cybersecurity Education: A CTF-based approach to engaging students in software security laboratories. Proceedings of the Canadian Engineering Education Association (CEEAA). <https://doi.org/10.24908/pceea.2023.17071>
- [65] Taylor, B., & Kaza, S. (2016). Security Injections@Towson: Integrating Secure Coding into Introductory Computer Science Courses. ACM Trans. Comput. Educ., 16(4). doi:10.1145/2897441
- [66] Nocera, S., Romano, S., Francese, R., & Scanniello, G. (2024). Software engineering education: Results from a training intervention based on SonarCloud when developing web apps. Journal of Systems and Software, 112308. <https://doi.org/10.1016/j.jss.2024.112308>
- [67] Teiniker, E., Seuchter, G., & Farrelly, W. (2019). Engaging Part-Time students in software security by inductive learning. 2022 IEEE Global Engineering Education Conference (EDUCON), 45, 491–499. <https://doi.org/10.1109/educon.2019.8725029>
- [68] Lwin, K. S., Poskitt, C. M., Kyong, J. S., & Leonard Wong, L. Y. (2022). XSS for the Masses: Integrating Security in a Web Programming Course using a Security Scanner, <https://doi.org/10.1145/3502718.3524795>
- [69] Rahman, A., Shahriar, H., & Bose, D. B. (2021). How do students feel about automated Security static analysis exercises? 2021 IEEE Frontiers in Education Conference (FIE). <https://doi.org/10.1109/fie49875.2021.9637201>
- [70] Tao, L., Qian, K., Lo, D., Parizi, R. M., Wu, F., & Chu, B. (2018). Enhancing secure software development education through relevant active learning. SoutheastCon, 1–5. <https://doi.org/10.1109/secon.2018.8478813>
- [71] Rahman, M. M., Barek, M. A., Akter, M. S., Riad, A. K. I., Rahman, M. A., Shahriar, H., Rahman, A., & Wu, F. (2024). Authentic Learning on DevOps Security with Labware: Git Hooks To Facilitate Automated Security Static Analysis. 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 2418–2423. <https://doi.org/10.1109/compsac61105.2024.00388>

- [72] OWASP Juice Shop | OWASP Foundation. (n.d.). <https://owasp.org/www-project-juice-shop/>
- [73] Taber, K. S. (2017). The use of Cronbach's Alpha when developing and reporting research instruments in science education. *Research in Science Education*, 48(6), 1273–1296. <https://doi.org/10.1007/s11165-016-9602-2>
- [74] Mishra, P., Pandey, C. M., Singh, U., Gupta, A., Sahu, C., & Keshri, A. (2019). Descriptive statistics and normality tests for statistical data. *Annals of Cardiac Anaesthesia*, 22(1), 67. https://doi.org/10.4103/aca.aca_157_18
- [75] Nahm, F. S. (2016). Nonparametric statistical tests for the continuous data: the basic concept and the practical use. *Korean Journal of Anesthesiology*, 69(1), 8. <https://doi.org/10.4097/kjae.2016.69.1.8>
- [76] Hong, M., Carter, M., Kim, C., & Cheng, Y. (2023). Data exclusion in policy survey and questionnaire data: Aberrant responses and missingness. *Policy Insights from the Behavioral and Brain Sciences*, 10(1), 11–17. <https://doi.org/10.1177/23727322221144650>
- [77] Rosner, B., Glynn, R. J., & Lee, M. T. (2005). The Wilcoxon signed Rank test for paired comparisons of clustered data. *Biometrics*, 62(1), 185–192. <https://doi.org/10.1111/j.1541-0420.2005.00389.x>
- [78] Sullivan, G. M., & Feinn, R. (2012). Using effect size—or why the P value is not enough. *Journal of Graduate Medical Education*, 4(3), 279–282. <https://doi.org/10.4300/jgme-d-12-00156.1>
- [79] Schafeitel-Tähtinen, T., Koskinen, J., & Helenius, M. (2024). Measuring cybersecurity teaching: Case university students in Finland. *International Journal of Learning and Teaching*, 10(4). <https://doi.org/10.18178/ijlt.10.4.481-490>
- [80] vm2 vulnerabilities | Snyk. (n.d.). Find Detailed Information and Remediation Guidance for Vulnerabilities and Misconfigurations. <https://security.snyk.io/package/npm/vm2>
- [81] NVD - CVE-2023-4806. (n.d.). Retrieved April 20, 2025, from <https://nvd.nist.gov/vuln/detail/CVE-2023-4806>
- [82] Nadeem, A. (2023). Cybersecurity as a crosscutting concept across an undergrad computer science curriculum: an experience report. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2310.07625>
- [83] Ajzen, I. (1991). The theory of planned behavior. *Organizational Behavior and Human Decision Processes*, 50(2), 179–211. [https://doi.org/10.1016/0749-5978\(91\)90020-t](https://doi.org/10.1016/0749-5978(91)90020-t)

- [84] Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, 84(2), 191–215. <https://doi.org/10.1037/0033-295X.84.2.191>
- [85] Bandura, A., & National Inst of Mental Health. (1986). *Social foundations of thought and action: A social cognitive theory*. Prentice-Hall, Inc.
- [86] Bell, R. S., Vasserman, E., & Sayre, E. C. (2015). Developing and Piloting a Quantitative Assessment Tool for Cybersecurity Courses. *American Society for Engineering Education-ASEE*. <https://doi.org/10.18260/p.23835>
- [87] Bishop, M., Ngambeki, I., Mian, S., Dai, J., Nico, P., Miloslavskaya, N., Drevin, L., Leung, W. S., & von Solms, S. (2021). Measuring Self-efficacy in Secure Programming. *Information Security Education for Cyber Resilience*, 615, 81–92. https://doi.org/10.1007/978-3-030-80865-5_6
- [88] Faklaris, C., Dabbish, L., & Hong, J. I. (2019). A self-report measure of end-user security attitudes (SA-6). (2019). *Proceedings of the 15th Symposium on Usable Privacy and Security, SOUPS 2019*, 61–77.
- [89] Flowerday, T., & Shell, D. F. (2015). Disentangling the effects of interest and choice on learning, engagement, and attitude. *Learning and Individual Differences*, 40, 134–140. <https://doi.org/10.1016/j.lindif.2015.05.003>
- [90] Hidi, S. (2006). Interest: A unique motivational variable. *Educational Research Review*, 1(2), 69–82. <https://doi.org/10.1016/j.edurev.2006.09.001>
- [91] Keller, J. (2010). *Motivational Design for Learning and Performance: The ARCS Model Approach* (1st ed. 2010.). Springer US. <https://doi.org/10.1007/978-1-4419-1250-3>
- [92] Kruger, H. A., & Kearney, W. D. (2006). A prototype for assessing information security awareness. *Computers & Security*, 25(4), 289–296. <https://doi.org/10.1016/j.cose.2006.02.008>
- [93] Loorbach, N., Peters, O., Karreman, J., & Steehouder, M. (2015). Validation of the Instructional Materials Motivation Survey (IMMS) in a self-directed instructional setting aimed at working with technology. *British Journal of Educational Technology*, 46(1), 204–218. <https://doi.org/10.1111/bjet.12138>
- [94] Pan, X. (2020). Technology Acceptance, Technological Self-Efficacy, and Attitude Toward Technology-Based Self-Directed Learning: Learning Motivation as a Mediator. *Frontiers in Psychology*, 11, 564294–564294. <https://doi.org/10.3389/fpsyg.2020.564294>
- [95] Colin, M., Bashir, M. N., & Memon, N. D. (2016). Self-efficacy in cybersecurity tasks and its relationship with cybersecurity competition and work-related outcomes. *USENIX Security Symposium*.

- [96] COMP.SEC.300: Secure Programming. (n.d.). Tampere Universities. Retrieved May 30, 2025, from <https://www.tuni.fi/studentsguide/curriculum/course-units/tut-cu-g-45737?year=2024>

APPENDIX A: SURVEY QUESTIONS

1. Survey Coding

This research uses pre- and post-surveys to analyze changes the participation in the exercise. For this reason, it is important to be able to link pre- and post-surveys. Create yourself a unique code, which is not directly associated to you as a person, so that you preserve your anonymity. It is also important that you remember this code because in post-survey, you need to use the same code. Below are steps that help you to generate and remember the code. Create a survey connecting code You find these rules in every survey, so you don't have to remember them. Use the following rules for the code:

1. Take the last letter of your mother's first name.
2. Take the last letter of your first name.
3. Sum the digits of the day and month of your birthday iteratively until you get a single digit.
4. Take the first letter of your street address.
5. Take the first letter of your birth city

Use rules in order (1, 2, 3, 4, 5). Example: for a person "Teemu Teekkari", who's mother is "Sari Lehtinen", address is "Kettukatu", birth city is "Muonio" and birthday is "23.11." - $> 23 + 11 = 34 \rightarrow 3 + 4 = 7$. Code with order (1, 2, 3, 4, 5): iu7km

2. Pre-Survey Questions

Section A: Demographics and Background

1. Age
2. Gender
3. What is your current role?
4. Years of Experience in Software Development
5. What is the highest degree or level of education you have completed?
6. Is information security or cybersecurity your major or the focus in your work?
7. How frequently do you work with CI/CD pipelines?
8. How familiar are you in using DevSecOps technologies (e.g., SAST, Docker)?

Section B: Knowledge with DevSecOps and Security Testing Tools

Please assess **the degree of how much you know about** them currently. In each case, make your choice in terms of what you know right now, not what you have known in the past or would like to know. Select "I can't assess this", if you can't assess your knowledge (I don't know anything about this, I know a little, I know about this, I know a lot about this, I know this thoroughly, I can't assess this)

1. The concept of integrating security into DevOps (DevSecOps).
2. Setting up and managing CI/CD pipelines using tools like Jenkins.
3. Static analysis tools (SAST), such as SonarQube, to identify code vulnerabilities.
4. Software composition analysis (SCA) tools (e.g., OWASP Dependency Check/ Snyk) to monitor third-party components.

5. File system scanning tools (e.g., Trivy) to ensure security in file repository.
6. Dynamic testing (DAST) tools (e.g., OWASP ZAP) to assess vulnerability in deployed applications.
7. Container scanning tools (e.g., Trivy) to ensure image security.
8. Building and deploying containerized application using docker.

Section C: Interest in DevSecOps and Security Testing Tools

The following items are the same as earlier. This time, please assess **the degree of how interested you are in** the following topics, concepts, and skills. In each case, make your choice in terms of how interested you are right now, not what you have been in the past or would like to be. Select "I don't know", if you can't assess your interest. (I'm not at all interested, I'm slightly interested, I'm interested, I'm very interested, I'm extremely interested, I can't assess this)

1. The concept of integrating security into DevOps (DevSecOps).
2. Setting up and managing CI/CD pipelines using tools like Jenkins.
3. Static analysis tools (SAST), such as SonarQube, to identify code vulnerabilities.
4. Software composition analysis (SCA) tools (e.g., OWASP Dependency Check/ Snyk) to monitor third-party components.
5. File system scanning tools (e.g., Trivy) to ensure security in file repository.
6. Dynamic testing (DAST) tools (e.g., OWASP ZAP) to assess vulnerability in deployed applications.
7. Container scanning tools (e.g., Trivy) to ensure image security.
8. Building and deploying containerized application using docker.

Section D: Self-Efficacy in Secure CI/CD Practices

Each statement below describes a task in DevSecOps. Please assess **the degree of how confident you are** in performing them. Try to assess your confidence of performing the task, which might be different than your knowledge of the task. In each case, make your choice in terms of how you feel right now, not what you have felt in the past or would like to feel. Select "I can't assess this", if you can't assess your self-efficacy. (No confidence at all, Slightly confident, Confident, Very confident, Completely confident, I can't assess this)

1. Working with Linux operating system.
2. Integrating security practices into an automated CI/CD pipeline.
3. Identifying and resolving security vulnerabilities in code using SAST tools.
4. Using SCA tools to manage vulnerabilities in open-source components.
5. Using DAST tools to detect security issues in deployed applications.
6. Using container security scanning to secure deployment environments.
7. Assessing the vulnerability of an application using the artefacts generated by the pipeline.

Section E: Attitudes Toward Integrating Security into CI/CD

Each statement below describes how a person might feel about the use of security measures in CI/CD. Please **indicate the degree to which you agree or disagree** with each statement. In each case, make your choice in terms of how you feel right now, not what you have felt in the past or would like to feel. There are no wrong answers. Select "I don't know", if you can't assess your attitude. (Strongly disagree, Somewhat disagree, Neither disagree nor agree, Somewhat agree, Strongly agree, I don't know)

1. Integrating security testing into the CI/CD pipeline is essential for producing secure software.
2. I believe that "shifting security left" (early detection) reduces overall development costs.
3. Automating security tests in CI/CD pipelines improves software delivery efficiency.
4. Security should be a shared responsibility among developers, operations, and security teams.
5. I am motivated to incorporate security practices into my development workflows.

Section F: Expectations for the Exercise

Each statement below describes your expectations for this exercise. Please indicate the degree to which you agree or disagree with each statement. In each case, make your choice in terms of how you feel right now, not what you have felt in the past or would like to feel. There are no wrong answers. Select "I don't know", if you can't assess your attitude. (Strongly disagree, Somewhat disagree, Neither disagree nor agree, Somewhat agree, Strongly agree, I don't know)

1. I expect that this exercise will improve my technical skills in implementing secure CI/CD pipelines.
2. Participating in this exercise will increase my confidence in using security testing tools.
3. I anticipate that the exercise will provide practical insights that I can apply to real-world projects.
4. I am optimistic that the knowledge acquired during this session will motivate me to explore further learning opportunities in secure CI/CD practices.
5. Overall, I expect that this exercise will be beneficial for my professional development.

Section G: Feedback

In case you have any comments about your answers or feedback to the survey, please write these here.

3. Post-Survey Questions

Section A: Demographics and Background

1. Age
2. Gender
3. What is your current role?
4. What is the highest degree or level of education you have completed?
5. Is information security or cybersecurity your major or the focus in your work?
6. Are you familiar with CI/CD pipelines? After this exercise.
7. How relevant was this exercise to your current work or future career goals?
8. How familiar are you in using DevSecOps technologies or tools after this exercise?

Section B: Knowledge with DevSecOps and Security Testing Tools

After the exercise, please assess **the degree of how much you know about** them currently. In each case, make your choice in terms of what you know right now, not what you have known in the past or would like to know. Select "I can't assess this", if you can't assess your knowledge (I don't know anything about this, I know a little, I know about this, I know a lot about this, I know this thoroughly, I can't assess this)

1. The concept of integrating security into DevOps (DevSecOps).
2. Setting up and managing CI/CD pipelines using tools like Jenkins.
3. Static analysis tools (SAST), such as SonarQube, to identify code vulnerabilities.
4. Software composition analysis (SCA) tools (e.g., OWASP Dependency Check/ Snyk) to monitor third-party components.
5. File system scanning tools (e.g., Trivy) to ensure security in file repository.
6. Dynamic testing (DAST) tools (e.g., OWASP ZAP) to assess vulnerability in deployed applications.
7. Container scanning tools (e.g., Trivy) to ensure image security.
8. Building and deploying containerized application using docker.

Section C: Interest in DevSecOps and Security Testing Tools

The following items are the same as earlier. This time, please assess **the degree of how interested you are in** the following topics, concepts, and skills after the exercise. In each case, make your choice in terms of how interested you are right now, not what you have been in the past or would like to be. Select "I don't know", if you can't assess your interest. (I'm not at all interested, I'm slightly interested, I'm interested, I'm very interested, I'm extremely interested, I can't assess this)

1. The concept of integrating security into DevOps (DevSecOps).
2. Setting up and managing CI/CD pipelines using tools like Jenkins.
3. Static analysis tools (SAST), such as SonarQube, to identify code vulnerabilities.
4. Software composition analysis (SCA) tools (e.g., OWASP Dependency Check/ Snyk) to monitor third-party components.
5. File system scanning tools (e.g., Trivy) to ensure security in file repository.
6. Dynamic testing (DAST) tools (e.g., OWASP ZAP) to assess vulnerability in deployed applications.
7. Container scanning tools (e.g., Trivy) to ensure image security.
8. Building and deploying containerized application using docker.

Section D: Self-Efficacy in Secure CI/CD Practices

Each statement below describes a task in DevSecOps. After doing the exercise, please assess **the degree of how confident you are** in performing them. Try to assess your confidence of performing the task, which might be different than your knowledge of the task. In each case, make your choice in terms of how you feel right now, not what you have felt in the past or would like to feel. Select "I can't assess this", if you can't assess your self-efficacy. (No confidence at all, Slightly confident, Confident, Very confident, Completely confident, I can't assess this)

1. Working with Linux operating system.
2. Integrating security practices into an automated CI/CD pipeline.
3. Identifying and resolving security vulnerabilities in code using SAST tools.
4. Using SCA tools to manage vulnerabilities in open-source components.
5. Using DAST tools to detect security issues in deployed applications.

6. Using container security scanning to secure deployment environments.
7. Assessing the vulnerability of an application using the artefacts generated by the pipeline.

Section E: Attitudes Toward Integrating Security into CI/CD

Each statement below describes how a person might feel about the use of security measures in CI/CD. After the exercise, please **indicate the degree to which you agree or disagree** with each statement. In each case, make your choice in terms of how you feel right now, not what you have felt in the past or would like to feel. There are no wrong answers. Select "I don't know", if you can't assess your attitude. (Strongly disagree, Somewhat disagree, Neither disagree nor agree, Somewhat agree, Strongly agree, I don't know)

1. Integrating security testing into the CI/CD pipeline is essential for producing secure software.
2. I believe that "shifting security left" (early detection) reduces overall development costs.
3. Automating security tests in CI/CD pipelines improves software delivery efficiency.
4. Security should be a shared responsibility among developers, operations, and security teams.
5. I am motivated to incorporate security practices into my development workflows.

Section F: Reflections on the Exercise and Learning Outcomes

Each statement below describes your experience for this exercise. Please indicate the degree to which you agree or disagree with each statement. In each case, make your choice in terms of how you feel right now, not what you have felt in the past or would like to feel. There are no wrong answers. Select "I don't know", if you can't assess your attitude. (Strongly disagree, Somewhat disagree, Neither disagree nor agree, Somewhat agree, Strongly agree, I don't know)

1. Participating in this exercise has improved my technical skills in implementing secure CI/CD pipelines.
2. This exercise has increased my confidence in using security testing tools.
3. The exercise has provided practical insights that I can apply to real-world projects.
4. The knowledge acquired during this exercise will motivate me to explore further learning opportunities in secure CI/CD practices.
5. I believe this exercise was beneficial for my professional development.
6. I am satisfied with the exercise as an introduction to secure CI/CD practices.
7. I would recommend this exercise to others who are new to secure CI/CD practices.

Section G: Open-Ended Feedback

1. What sections of the exercise did you find most helpful?
2. How the exercise could be improved? Any thoughts.
3. What are your key takeaways from this exercise?
4. Is there anything else you would like to share about your experience with this exercise or the survey?