DevSecOps: Enabling Security by Design in Rapid Software Development

Amr Mustafa Ahmed

a.m.a.a.ahmed@uu.nl

First Supervisor
Dr. V. Moonsamy
v.moonsamy@uu.nl

Second Supervisor
Dr. S. Overbeek
s.j.overbeek@uu.nl

Daily Supervisor
C. Manadis
chris.manadis@accenture.com

January 30, 2019

Universiteit Utrecht

accenture

Abstract

Context: *Organizations have been using DevOps for several years now to enable faster delivery of software to the market. End-to-end DevOps is becoming the goal of organizations. Within this rapid development, security becomes a concern. Security has always been a separate silo that defines security requirements and demands for certain security controls to approve new software code. In most cases, security is involved at a later stage when it is expensive to make changes or apply fixes. Thus, it becomes an added layer on top of the application, rather than an integrated part.* **Objective:** *The goal of the research is to integrate security concepts in the development and the operation phases of software production. This includes understanding the meaning of application security and the risks that can be mitigated during the DevOps process.* **Method:** *The main method used in this research is Design Science Research as defined by Hevner and Chatterjee [55]. Further, the following techniques were used; systematic literature review, interview, case study and focus group. The research starts by understanding the problem and the context where this problem emerges from. For this step, a literature review and a round of interviews took place. This was followed by a design science cycle where a solution of the problem is developed and improved based on the knowledge collected in the previous step. The two created artifacts resulted from a case study. To validate the results, a focus group session was planned where experts gave their feedback on the artifacts and the final artifacts are created. The results are reported in this thesis report.* **Results:** *This research results in two main artifacts, first the impact model. This model shows the main areas that will be touched when trying to implement security within a DevOps team. The second artifact is the DevSecOps framework that illustrates what security measures can be introduced in DevOps pipeline.* **Conclusion:** *It is possible to include security within a DevOps team. That will require the team to learn a new set of skills and to add new tools to the pipeline. It is cheaper to patch security issues as early as possible, therefore focusing on security has to happen in the planning phase of each increment.*

Keywords: *DevOps, DevSecOps, DevOps Security, Software Security, Application Security, Time to Market, Software Production.*

List of Figures

1.1 OSI Model 5

2.1 Research Questions Structure 9

3.1 Design Science Research, based on Henver [56] 12
3.2 Linear Iterative Activities 13
3.3 Design Science Research in The Context of This Research . . . 16
3.4 Case Study Design based on Yin [106] 19
3.5 Case Study Analysis Tree of Nodes 21

4.1 DevOps Stages based on Rathod and Surve [83] 28
4.2 DevOps Outline 29
4.3 DevOps in Practice 38
4.4 DevSecOps in Literature and Practice 40

5.1 Impact Areas of DevSecOps 42

6.1 DevSecOps Initial Framework 49
6.2 OWASP Risk Classification 52
6.3 Risk Classification Action List 53
6.4 Penetration Testing Steps [37] 60

7.1 DevSecOps Impact Areas 68
7.2 DevSecOps Proposed Framework 69

C.1 DevOps Stages based on Rathod and Surve [83] 101

List of Tables

3.1 Systematic Literature Review Results 17
3.2 Round 1 Interviewees List 18
3.3 Round 2 Interviewees List 20
3.4 Focus Group Participants List 22

4.1 Software Security Principles [38] [19] [87] 30

Contents

1 Introduction **1**
1.1 This Research . . . 1
1.2 Problem Statement . . . 3
1.3 Scope . . . 4
1.4 Relevance . . . 4
1.5 Structure . . . 6

2 Research Questions **7**
2.1 Main Research Question . . . 7
2.2 Sub-Research Questions . . . 8
2.3 Relation between sub questions and main question . . . 9

3 Research Approach **11**
3.1 Research Method . . . 11
3.2 Design Science Research . . . 13
3.2.1 Environment . . . 13
3.2.2 Design Science . . . 14
3.2.3 Knowledge Base . . . 14
3.2.4 Rigor Cycle I . . . 14
3.2.5 Relevance Cycle I . . . 14
3.2.6 Design Cycle . . . 15
3.2.7 Rigor Cycle II . . . 15
3.2.8 Relevance Cycle II . . . 15
3.3 Literature Review . . . 16
3.4 Additional Literature . . . 17
3.5 Interviews . . . 17
3.6 Case Study Design . . . 18
3.6.1 Planning . . . 19

3.6.2 Interviews 20
3.6.3 Analysis 21
3.7 Focus Group 21

4 Background 23
4.1 Literature Review 23
4.1.1 Agile and Software Production 23
4.1.2 DevOps 24
4.1.3 DevOps Practices 25
4.1.4 DevOps Stages 27
4.1.5 Security in Software Production 29
4.1.6 Security Testing 34
4.1.7 DevSecOps 35
4.2 Interviews 37
4.2.1 DevOps in Practice 37
4.2.2 Security in Practice 39
4.3 DevSecOps in Theory and Practice 40

5 Impact Areas of DevSecOps 41
5.1 Towards The Impact Area Model 41
5.2 People 41
5.2.1 Culture 42
5.2.2 Team 43
5.2.3 Ownership 44
5.2.4 Accountability 44
5.2.5 Learning Process 44
5.3 Tools 45
5.3.1 Automation 45
5.4 Values 45
5.4.1 Measurement 45
5.4.2 Velocity 46
5.4.3 Control 47
5.5 Processes 47

6 DevSecOps Initial Framework 48
6.1 Initial Baseline Security 48
6.2 Planning Board 50
6.2.1 Security Requirements 50

6.2.2 Security Classification . . . 51
6.2.3 Threat Modeling . . . 53
6.2.4 Security Architecture Review . . . 53
6.3 Development Environment . . . 54
6.3.1 Secure Coding . . . 54
6.3.2 Coding Guidelines . . . 55
6.3.3 Static Validation . . . 56
6.3.4 Dynamic Testing . . . 56
6.4 Testing Environment . . . 57
6.4.1 Vulnerability Scanning . . . 58
6.4.2 Fuzz Testing . . . 58
6.4.3 Penetration Testing . . . 58
6.5 Production Environment . . . 59
6.5.1 Security Monitoring . . . 61
6.5.2 Configuration Check . . . 62
6.5.3 Traceability . . . 62
6.6 Infrastructure Security . . . 63
6.6.1 Environment Segregation . . . 63
6.6.2 Vulnerability Mapping . . . 64
6.6.3 Patching and Update Management . . . 64
6.6.4 Access Management . . . 65
6.6.5 System-wide Traceability . . . 65
6.6.6 Encryption . . . 65

7 Validation and Improved Artifacts 67
7.1 Impact Areas . . . 67
7.2 DevSecOps Proposed Framework . . . 67

8 Discussion and Limitations 72
8.1 Discussion . . . 72
8.2 Limitations . . . 73
8.2.1 Construct Validity . . . 73
8.2.2 External Validity . . . 74
8.2.3 Reliability . . . 74
8.2.4 Evaluation . . . 74

9 Conclusion 75
9.1 Sub-Research Question 1 Answer: 75
9.2 Sub-Research Question 2 Answer: 76
9.3 Sub-Research Question 3 Answer: 76
9.4 Sub-Research Question 4 Answer: 77
9.5 Main Research Question 77

10 Research Contribution and Future Work 79
10.1 Research Contribution 79
10.2 Future Work 80

Appendices 93

A Literature Review Protocol 94

B Interview Protocol 96

C Case Study Protocol 99

D Informed Consent 103

Acknowledgment

"If you try, you risk failure. If you don't, you ensure it."

It is about two years now and I can't believe I am already finishing this master. What a journey! Now that I am at the finishing line, I look back at the things I learned and the people I met and I feel proud that I finished. Very challenging was this master, coming from a totally different educational background, language and culture. Now, I am definitely more confident about my knowledge and my capabilities and ready for future.

Just like most of the students in the thesis phase, I had my own doubts about the topic, research method or even myself. However, I was lucky that I always had someone to ask or to talk to. Veelasha, Sietse, Chris, Arno and Burhan, thank you very much for bringing the best in me, this wouldn't have been possible if it wasn't for you, thank you for guiding me and pointing me to the right direction. To all my teachers at Utrecht University and to every single person who participated in my thesis, thank you. To my friends, Klea, Baharak, Oka and the international group, big fat thank you for being there through thick and thin, you made this master easier. I carry now memories that I will never forget. Thanks to Robin, Thana, Faris, Abdulrahman and Yannick who contributed directly or indirectly to my thesis, I appreciate the support. I want to extend this thank-you note to Accenture for offering me this amazing experience and to the GDPR interns and employees. Lastly, my biggest thanks is to my family, mom, dad, brother and sisters no words can express the gratitude I feel for your ultimate support all the way.

Amr Mustafa
January 14th, 2019
Accenture - Utrecht
The Netherlands

Chapter 1

Introduction

This chapter provides a brief introduction on the concept of software development. Thereafter specifying the scope, relevance, and overall structure of the research.

1.1 This Research

Software development has changed dramatically in the past two decades. New development languages, technologies and computing power have been introduced and improved during the course of the years. Development methodologies are another driver of this development movement. Moving from Waterfall-driven development [5] to Agile [6] enabled developers to address market needs more precisely in reasonable time frames, and with a possibility to iterate constantly as long as needed [6]. Further, DevOps was introduced to leverage even faster development and deployment [71] [28]. DevOps is based on the collaboration of two teams, Development and Operations, becoming one, allowing both teams to communicate better and work together to solve problems. Additionally, DevOps fosters automation where possible to enable faster cycles of operations and development, allowing organizations to deliver value sooner to clients.

Software taxonomy includes a variety of software production and operation models based on many variables such as market [41], development environment [24], production environment [22] and licensing [22]. The value delivered to the customer is highly based on these variables. For example, in a cloud-based software, it is easier to rollout updates than a client-based software. Updating cloud-based software requires installing updates on the cloud and users can immediately access these updates. Whereas in a client-based software, updates need to be installed on each client's device separately. Another example is the licensing models. Software is considered as a *service* when a subscription model is followed [49], while it is a *product* when users are granted licenses. Many other models do exist to classify software. Within all these alternatives, DevOps operates to deliver value faster to clients regardless of the end deployment environment or licensing model. Clients get fixes, updates and new features to their application as soon as they are ready to be shipped.

As exciting as this sounds, security can become a major concern; data breaches, account hijacking, malicious insiders to name a few [92]. Developing reliable software that can minimize these threats requires intensive security tests. In a DevOps environment, rapid changes and updates are deployed automatically into the production environment. Tens or even hundreds of updates can be deployed everyday. In this hectic race of updates, security might not be handled properly to keep up with the delivery timeline. Normally, security is a slow process that requires time and resources, therefore, slowing down the whole DevOps process, or they are simply overlooked leaving not only the software at risk, but also all the clients' data stored or processed using the application. In practice, security tests, if exist, take place at the testing stage. All security tests, as well as other tests are conducted only during that stage. However, each stage in DevOps has its own environment and therefor requires different tests.

DevSecOps is a new concept used for embedding the security perspective into DevOps processes and embrace continuous security during all of the DevOps stages. Continuous security was described by Fitzgerald and Stol as:

> "*transforming security from being treated as just another non-functional requirement to a key concern throughout all phases of the development life cycle and even post deployment, supported by a smart and lightweight approach to identifying security vulnerabilities*" [78].

DevSecOps goes beyond providing tools, roles and activities to changing the development and operations culture to ensure a higher level of security. DevSecOps is not a final state, it is a journey.

1.2 Problem Statement

DevOps enables software to be delivered faster with smaller cycles called "*increments*" or "*iterations*". This well-established pace of producing software is not expected to slowdown. It might rather get faster and faster. At the same time, applications are expected to have robust security against threats. They are also expected to recover quickly when security measures fail to mitigate an attack. In a traditional working scheme, security tests are performed after the development of code is concluded in so-called "*Testing Environment*" or even later in the "*Acceptance Environment*". Once an increment passes the tests, it is labeled "*secure*" and shipped to customers, or deployed to the production environment. This approach has several issues:

- **Agility**: Security tests require time. A tester needs to create test cases and create a testing environment to prepare for the tests. If an increment does not pass a test, the developers are notified and fixes are applied to the increment. Afterwards, the tester creates new cases and new environment to test again. This preparation for tests take a long time when performed manually. Thus delaying the deployment stage and clients can not receive the latest updates.

- **One-time tests**: Once an increment is labeled secure, it is deployed into the production environment. However, no follow-up tests take place, although new vulnerabilities are disclosed everyday. These one-time tests do not guarantee that software is secure all the time. Which leaves the whole application and the data it contains is serious risk.

- **Security silos**: Security is always the responsibility of the security team. This team works in isolation of the other teams and with its own pace. At the same time, the security team takes full responsibility in case of a security breach. This situation results in misalignment between the security team who feels the pressure of securing the application and the DevOps team who demands agility.

- **An afterthought**: when security is not considered at the start of an increment, it is not an integrated part of the software. Security becomes an added layer that is placed afterwards to only meet a regulation or a policy. Security is considered too late in the process that it results in high costs when applied.

The way of doing security has to be adapted to fit in this rapid DevOps cycle to ensure the safety of clients, their data and the application. One data breach can be very expensive for everyone and a real threat for the organization's existence. Based on the aforementioned reasons, the problem statement is stated as follows: "*Software security is a slow process that hinders rapid development in a DevOps framework. Yet, it is a critical requirement to ensure the safety of the application and the clients' data.*"

1.3 Scope

DevOps as a method is widely adopted in the software market. It is flexible in a way that each organization has its own implementation of DevOps. Security on the other hand, from an application security perspective, is influenced by the type of the application and the production environment. For example, applications that run internally on a closed network are less likely to be vulnerable than those which work on cloud.

The scope of this research has two dimensions, 1) Application development based on DevOps and 2) Security. The direction of this research is aiming for applications that are developed using DevOps in particular. This includes, but not limited to, web based applications, cloud applications and server-client applications. On the security dimension, security is considered at the application layer. The International Organization for Standardization (ISO) has developed a conceptual model named Open Systems Interconnection (OSI) that describes the conceptual levels of communication between systems [63]. The model, shown in Figure 1.1, consists of 7 layers. Attacks can happen at any of these layers. For example, on the network layer (layer 3), an attacker can target a router with a Black Hole attack [18] resulting in permanent packet loss. This research focuses on the attacks that happen on the application layer (layer 7) which are directly connected to software development.

1.4 Relevance

Security is always looked at as an expensive and time consuming process that is slowing down the development of software and works against the main concept of DevOps of rapid cycles and immediate implementation of

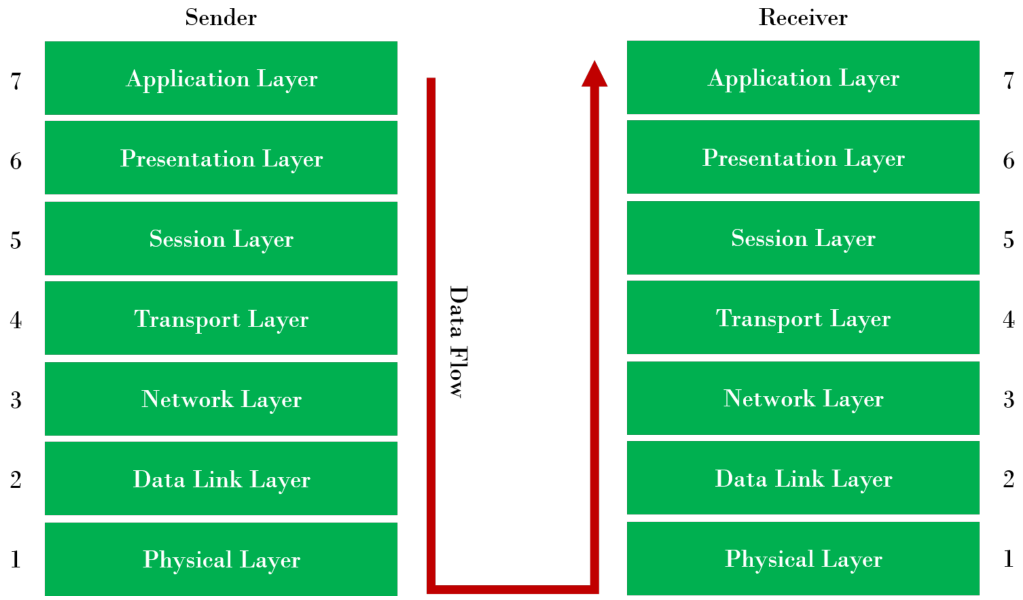


Figure 1.1: OSI Model

fixes, updates and new features. The idea of including security at an earlier stage of the process helps reduce the overhead time by doing the right things in the first place. Eventually, all of these processes and measures aim to comply with security standards and local regulations to protect users and their data.

DevSecOps is a new concept derived from DevOps. While DevOps focuses on Development and Operations of software, DevSecOps adds a new team to embrace security in this rapid process [76]. The need for the DevSecOps concept emerged from the community working and adopting DevOps. Gartner is expecting that more than 80% of rapid development teams will use DevSecOps practices by 2021 [44]. This number is driven by high adoption rate from security vendors who are developing security solutions that take DevOps agility into account. The concept of including security within DevOps has many keywords such as “DevSecOp, SecDevOps, DevOpsSec, Secure DevOps, SecOps and rugged DevOps” [45]. Although the industry has developed a foundation of the DevSecOps concept, the scientific research is still *very* limited.

1.5 Structure

The following chapter, chapter 2, contains the main research question and the sub-research questions. Chapter 3 describes the research approach and the method used in this research. Additionally, it contains the scientific ground for each research instrument used such as literature review and case study. Chapter 4 contains the background information of the context of the research, that is the literature review and the interviews conducted to understand the research environment. Chapter 5 presents the impact areas of including security in DevOps teams, which is the first artifact. Chapter 6 presents the DevSecOps framework which is the second artifact. In chapter 7, the validated artifact are presented. It includes the main changes that has been introduced to the initial models. Chapter 8 has the discussion and the limitation of this research. The sub-research questions and the main research question are finally answered in chapter 9. The last chapter, chapter 10, presents the research contribution of this thesis and points out the following steps for future research.

Chapter 2

Research Questions

This chapter details the questions used to direct this research. There is one main research question that is broken down to four sub-research questions.

2.1 Main Research Question

The overall goal of the research is to integrate security concepts in the development and the operation phases of software production. In present, security is approached in the testing stage of an application and sometimes in production depending on the application. For both stages penetration testing as well as other tests and tools are used. However, security is not applied in all the stages, therefore the main research question consists of two core aspects. The first aspect is *continuous security* which indicates how security can be involved during the whole cycle of DevOps and not only limiting security to couple of control gates. The second aspect is *rapid value delivery* which refers to how new updates that comply with security policies are deployed to the production environment within DevOps time frame.

Main Research Question: ***How can security be added to a DevOps framework to guarantee continuous security and rapid value delivery?***

2.2 Sub-Research Questions

The main research question is broken down into four sub-research questions. The first one is a knowledge question while the other three are research questions.

RQ1: **How is security approached in DevOps framework in theory and in practice?**

The first sub-research question aims to position the research by exploring security in DevOps teams from two perspectives: theory and practice. In the theory part, the goal is to understand the scientific research around DevOps and the stages where security controls take place. In the practical part, the focus will be on how this theory is applied in real-life and what is the gap in between. Additionally, the theory used will create the base for the proposed artifacts.

RQ2: **What is the impact of implementing DevSecOps on organizations?**

The second sub-research question defines the main outline of DevSecOps. Introducing security into DevOps teams will bring changes to how the teams are formed. This question defines these areas so that they are highlighted during a transformation process.

RQ3: **What are the changes that DevSecOps introduces to a DevOps team and how will those changes impact a DevOps team?**

After answering this question, the next step is to create the DevSecOps framework. DevOps theory is used to create the outline of DevSecOps. The framework answers the questions of what will change in current DevOps *processes*. This question dives deeper to DevSecOps' stages and practices.

RQ4: **How are the proposed artifacts validated by experts?**

The last sub-research question is the validation question. The constructed artifacts are to be validated and improved in an iterative cycle. The answer of this question will include improvements of the constructed artifacts. The artifacts will be ready to be applied in real-life.

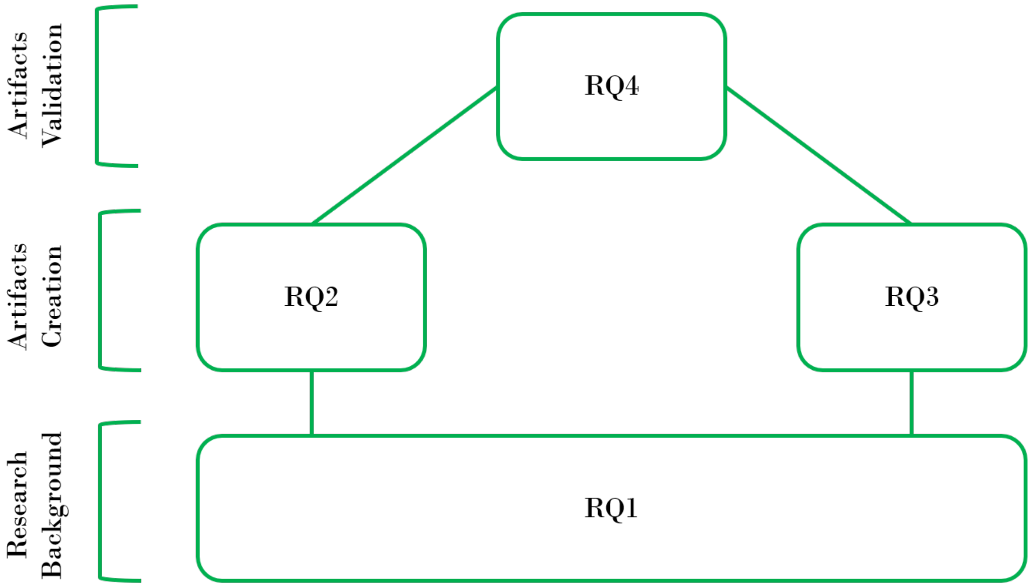


Figure 2.1: Research Questions Structure

2.3 Relation between sub questions and main question

The sub-research questions are divided in three groups. Figure 2.1 illustrates how the sub-questions relate to each other. The bottom layer in the Figure, which is RQ1, is mainly for creating a concrete background about the domain of this research. The answer to this question is directing and influencing the remainder of the research. The second layer includes the sub-research questions of constructing the artifacts, answering the questions of what and why. RQ2 is defining the main touch points of DevSecOps in an organization by giving a high abstraction about the main change areas. RQ3 focuses in detail on the processes of DevOps and what can be included from a security perspective on the DevOps processes. Finally, RQ4 is focused on evaluating these artifacts and explore the areas of improvements.

Each of the sub-research questions represents a part of the puzzle. While the first question is creating a theoretical and practical base for the research, the second and third questions aim to build theory on DevSecOps. While the second question elaborates on the DevSecOps' impact areas, the third question explains in details the framework of DevSecOps. The last piece of the puzzle is to validate how all these components hold together in the eyes of experts.

Chapter 3

Research Approach

This chapter describes the research method, the different cycles it had and the instruments used to conduct the research. It also includes the scientific grounds of each of the instruments and the reasons they were selected for this research.

3.1 Research Method

This research follows Design Science Research as defined by Hevner and Chatterjee [55]. Design Science Research is a solution-oriented method. While it provides solutions to problems in the real-world, it also aims to provide iterative solutions that relate to the environment and the existing knowledge around the problem. Design Science Research aims to carefully study the problem environment, create solutions for that context and finally apply and evaluate these artifacts [56] [81]. It is defined as a method that "*seeks to create innovations, or artifacts, that embody the ideas, practices, technical capabilities, and products required to efficiently accomplish the analysis, design, implementation, and use of information systems.*" [57].

Hevner has identified the processes involved in design science research [56]. Figure 3.1 illustrates these processes, referred to as "*cycles*". These cycles connect three main domains, **Environment**, which refers to the context where the problem has emerged from. This includes people and organization, IT systems and infrastructure. **Knowledge Base** is the reference point for all the knowledge that exists around the problem context including scientific research, the experience of the people working in that specific prob-

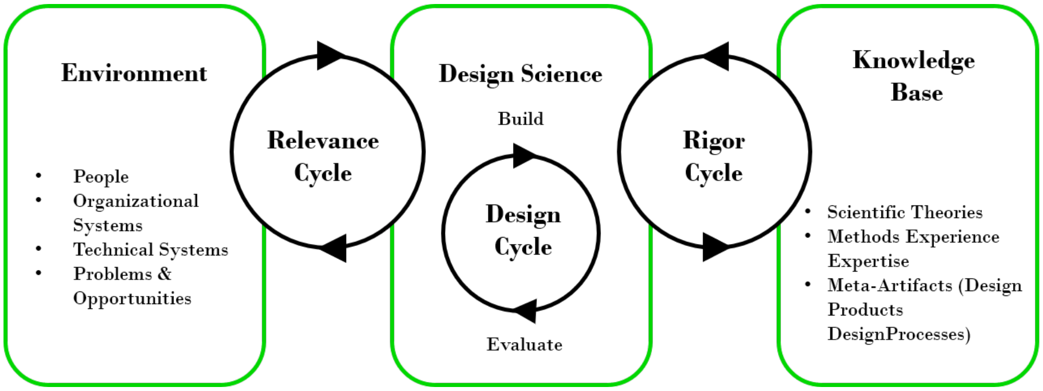


Figure 3.1: Design Science Research, based on Henver [56]

lem domain and current alternative solutions. The **Design Science** is the process of compiling the knowledge, or generating new knowledge, to solve the problem and validating the proposed solutions. It is an iterative process of building and validating how proposed solutions fit in the context. These three domains are connected by three main bonding cycles and each cycle has two main activities.

Relevance Cycle: in this cycle, there is a connection between the Environment and the Design Science. In the beginning of the research, Environment is considered as the input of the Design Science. In this step, researchers will collect requirements and build a context of the research, while towards the end of the research, the Environment receives the results of the research, the artifact, and applies it in the predefined context. The relevance cycles ensures that the resulted artifact do indeed solve the problems.

Rigor Cycle: on the other side, a rigor cycle aims to ensure the novelty of the proposed solution, connecting the design science with the knowledge base. First, the researchers have to ground the research by looking for the knowledge in the domain of the research. The knowledge includes scientific research, experts' opinion and existing solutions. Second, when a final solution is developed, this solution is added to the knowledge base and reused, where applicable.

Design Cycle: the last cycle is an internal cycle which aims to combine the knowledge with the environment and results in realistic solutions that are innovative and applicable to the predefined context. Like the previous cycles, the design cycle has also two activities building an artifact, and testing the

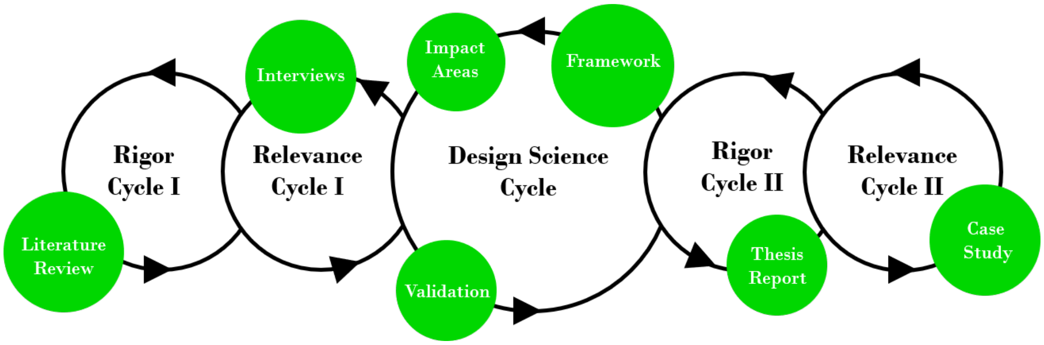


Figure 3.2: Linear Iterative Activities

artifact. In the building activity, researchers use all the knowledge collected from the knowledge base and all the requirements from the environment into account. Additionally, in the validation activity, the usefulness of the application is tested. This is a continuous cycle that happens until satisfactory results are reached.

3.2 Design Science Research

Figure 3.2 shows a linear sequence of iterative activities that are followed in this research. Mainly, the steps are categorized in five cycles, where each phase contains one or more activities. Each cycle connects two of the three domains. Figure 3.3 shows how the framework of Hevner is applied. The following sections elaborate in detail the three domains and the five cycles.

3.2.1 Environment

The Environment, where this research took place, was Accenture. A large consulting company with more than 470,000 employees around the world. Accenture is a leading company in technology and innovation market and they have strategic partnerships with Amazon, Microsoft and other high tech companies. As a consulting company, Accenture's employees work at a client to solve a problem. In this research context, participants worked at a wide variety of sectors on software delivery projects.

3.2.2 Design Science

In the Design Science domain, the artifacts were created based on the data collected from the other two domains. Further, they were validated with the daily supervisor. Thereafter, the artifacts were formally validated on a focus group session.

3.2.3 Knowledge Base

The Knowledge Base used for this research was mainly literature and experts. The literature review was based on a systematic literature review and that is detailed in section 3.3. Additional literature sources were used to support the ideas proposed in the interviews. For these supporting material, google scholar was the main source. Finally, this thesis report is created and published in the university thesis library.

3.2.4 Rigor Cycle I

In this phase, one main activity was performed, which was literature review. The main task of this activity was to look up and summarize the current knowledge of DevSecOps, DevOps and Software Security. Performing this task requires collecting scientific papers from multiple search engines and reading these papers to understand the state of the art. The detailed protocols are reported in appendix A. The results and findings are reported in the literature review section 4.1.

3.2.5 Relevance Cycle I

The aim of this first cycle of relevance is to understand the context of the problem in practice. The focus is on the use of DevOps in software production and security. Running this cycle includes interviews with experts in software production, developers, testers and security experts.The interview protocol of this interview is in appendix B. The results of this activity is reported in section 4.2.

3.2.6 Design Cycle

Both of the previous phases are used as inputs in this core phase. This phase results in the main output of the research. Two main activities are planned: build artifacts and validate artifacts. In the build activity, two main outputs are expected.

- **Impact Areas:** is an abstract model that represents the main touch points that DevSecOps interacts with in the process of transforming to DevSecOps. This model is introduced in section 5

- **DevSecOps Framework:** is a detailed view of the changes that happened within the DevOps process. The process itself does not change, however, the activities that happen within this process encounters introducing security measures. The DevSecOps framework is introduced in section 6.

After creating these two artifacts, they were improved with the daily supervisor. Afterwards, they were validated and improved in a focus group. The attendees were composed of all DevSecOps teams namely: Development, Operations and Security to provide a balanced validation. The artifacts were improved afterwords.

3.2.7 Rigor Cycle II

Once these artifacts are built, validated and improved, they are reported in this thesis report. These deliverables contribute to the scientific body and add to the knowledge base. The artifacts are then available for reuse in relevant contexts.

3.2.8 Relevance Cycle II

The main idea of this research was to solve a problem in real-life scenario. The ideal situation is to apply the results in real-life, however applying the results in real-life requires a long time to adopt a change, therefore it is out of the scope of this research, yet, it is recommended to apply the results in real-life and observe changes.

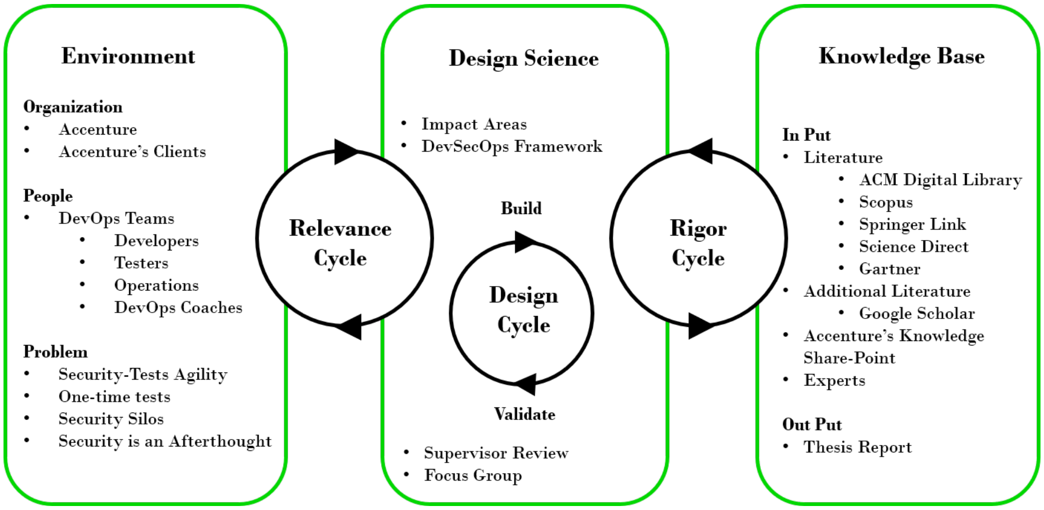


Figure 3.3: Design Science Research in The Context of This Research

3.3 Literature Review

The literature section in this research follows a systematic literature review. Kitchenham [66] defined systematic literature review as a "*means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest.*" This method was selected because of two main reasons, first, a systematic literature review is used to show a gap in research by summarizing current research work [12]. Second, it is used to form the base of the following steps of the research [66]. In this research, the goal is to identify the research done on DevSecOps and verify the research gap. Additionally, the review will be used as a base for the rest of the research and artifacts.

Three main phases are identified in Evidence-Based method by Kitchenham et al. [67] and explained by Brereton et al. [10]. A *review protocol* is first created [12] [10]. The protocol, which is attached in appendix A, includes the strategy of the research such as the search engines, keywords used and the limit of search results. Additionally, it contains the inclusion and exclusion criteria of the literature found. Second, the literature review is conducted and the search results are shown in table 3.1. The collected papers gave a good foundation of the topic and the research, however, in some cases, additional sources were used. These additional resources do not necessary comply

Item	**Number of Papers**
Total number of papers collected	161
Total excluded papers	61
Excluded because of: Not English	4
Excluded because of: Duplication	6
Excluded because of: Older then 2014	3
Excluded because of: Book	8
Excluded because of: Irrelevant	40

Table 3.1: Systematic Literature Review Results

with systematic literature review exclusion criteria. Third, the results of the literature review are reported in section 4.1.

3.4 Additional Literature

The systematic literature review resulted in primary sources. However, additional literature was required to explain the components of the artifacts. Snowballing within systematic literature review is explained by Wolin [103]. This method was followed partly. First, the "*Start Set*" was the literature found in the systematic literature review. The "*Iterations*" refers to the direction of finding additional sources which can be Backward Snowballing or Forward Snowballing. The backward snowballing means looking into the references used in the primary sources to find additional sources, while the forward snowballing means finding the literature which cite the primary sources. In this research, backward snowballing was used because, the domain of this research is new and forward snowballing does not result in relevant literature. Finally, for the "*Data Extraction*" the sources were scanned and relevant information was reported.

3.5 Interviews

Interviews are conducted in two phases of the research. First in the relevance cycle it is used to understand the context of the problem, which is then used again in the design science cycle to validate the constructed artifacts. The second round of interviews is detailed in the case study section

Interviewee ID	Code	Role	Years of Experience
Interviewee 001	int.01	Tester	3 Years
Interviewee 002	int.02	Developer	10 Years
Interviewee 003	int.03	DevOps Coach	6 Years
Interviewee 004	int.04	Developer	3 Years

Table 3.2: Round 1 Interviewees List

3.6.2. Interviews are one of the main tools used in collecting data in qualitative research [90]. There are three main types as defined by Doody et al. [27]: structured, unstructured and semi structured. For this research, semi structured interviews were conducted. This type of interviews allows researchers to ask additional follow-up questions for more clarification, which in return enrich the quality of the interview and the research. The main protocol is created for each phase. The interviews are recorded, transcribed and analyzed using Nvivo, a software used for qualitative research.

The first round of interviews was mainly focused on understanding how security is applied in real DevOps projects. For this stage 4 interviewees were asked about DevOps and Security. The full interview protocol is enclosed in the appendix B. Table 3.2 shows details about the interviewees and the code name that will be used for referring to them in the remainder of this thesis. The interviewees were asked to sign an informed consent that is enclosed in appendix D.

3.6 Case Study Design

The case study is conducted to answer RQ2 and RQ3. It was designed based on the description explained by Wohlin et al. [104] and Yin [106]. They defined case studies as "*an empirical method aimed at investigating contemporary phenomena in their context*" [104]. Additionally, they emphasized the use of multiple sources of evidence. A case study might contain many other research method elements such as interviews, surveys and experiments. The results of the case study do not provide statistical significance, but rather makes conclusions based on documents, figures and statements. The following sections elaborate on the case study design in this research.

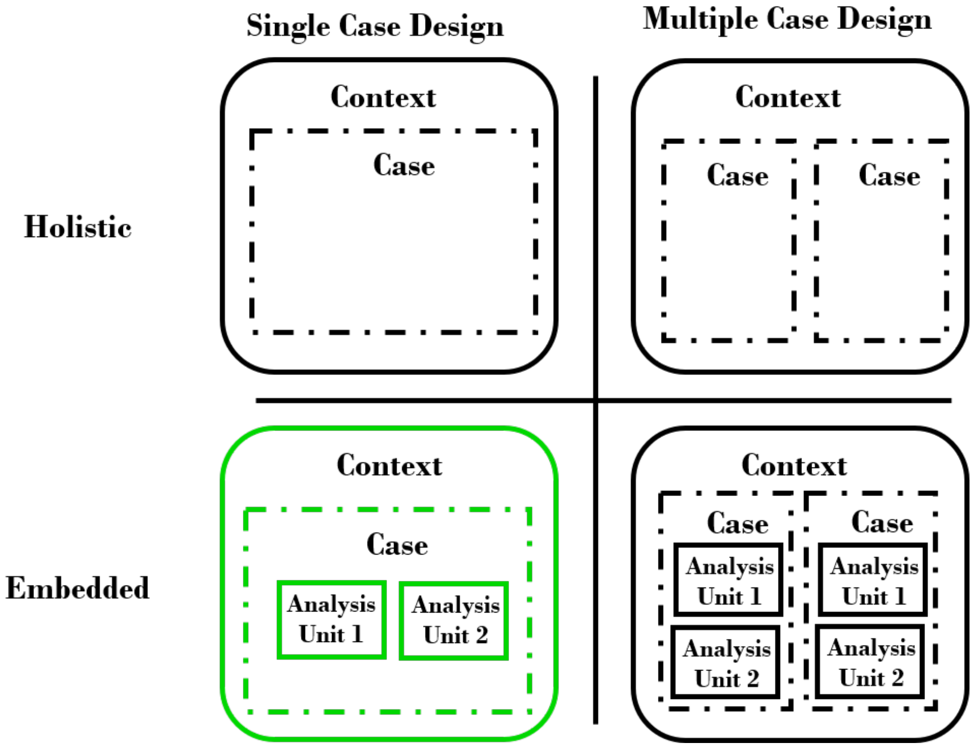


Figure 3.4: Case Study Design based on Yin [106]

3.6.1 Planning

A case study can be classified in one of four categories as demonstrated by Yin [106]. Figure 3.4 illustrates that a case study can have a holistic or embedded view. On another level, a case study can be performed on one case or multiple cases. This research is applying an embedded analysis on a single case design, which is highlighted in green in Figure 3.4. **The case** is DevOps teams and the analysis units are Development team, Operations team and Security team in the context of rapid software production. The **objective** of the case study is to create an impact model as well as a DevSecOps framework. The case does not particularly focus on one team rather on the individuals in the teams and the process of applying DevOps stages. The **theory** used to define the outline of the case study is the DevOps theory and the theory of software security. A systematic literature review is conducted to define this outline and it is presented in section 4.1. This planned case study aims to answer **sub-research questions** two and three that are presented in section 2. The **methods** used in this case study are interviews, literature, security strategy documentations, standards and white papers. Finally, the **selection strategy** consists of grouping content in three groups: Development, Operations and Security, and conclusions will be based on making a triangulation. This technique is used to cover all views of the case. The case study protocol is enclosed in appendix C.

Interviewee ID	**Code**	**Role**	**Years of Experience**
Interviewee 005	int.05	Developer	3 Years
Interviewee 006	int.06	Software Architecture	10 Years
Interviewee 007	int.07	Tester	10 Years
Interviewee 008	int.08	Quality Engineer	7 Years
Interviewee 009	int.09	Application Security	8 Years
Interviewee 010	int.10	Application Security	8 Years
Interviewee 011	int.11	Security Operations	1 Years
Interviewee 012	int.12	Cybersecurity	3 Years
Interviewee 013	int.13	Operations	11 Years
Interviewee 014	int.14	Service Delivery	13 Years
Interviewee 015	int.15	Operations	12 Years
Interviewee 016	int.16	Containerization	1 Years

Table 3.3: Round 2 Interviewees List

3.6.2 Interviews

A second round of interviews is conducted as one of the data sources used. The second round had a different line of questions that are enclosed in the appendix C.5. The goal was to understand the main impact areas and also to identify the security interventions that can be taken in each step of the DevOps process. Interviewee selection was based on domain of experience and knowledge about DevOps and software security. Table 3.3 shows the interviewees who participated in this phase and the code names that will be used in the remainder of this research. The interviewees were asked to sign an informed consent that is enclosed in appendix D.

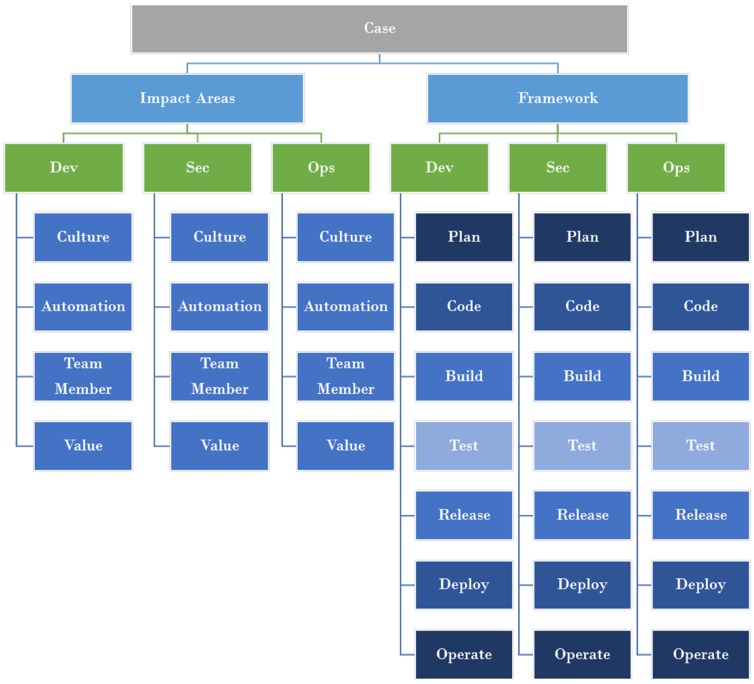


Figure 3.5: Case Study Analysis Tree of Nodes

3.6.3 Analysis

The Analysis of the case study required using Nvivo. In this application, a tree of nodes was created to contain all parts of the artifacts. Figure 3.5 shows all the nodes created in 3 levels of abstraction. The framework branch includes the perspectives of DevOps-stages in all roles, i.e. development, security and operation. The same stage is compared across the teams and conclusions are based on that. On the impact area model branch, four main components are used to determine the impact area model, Culture, Automation, Team and Value. After analysis, the initial artifacts were created.

3.7 Focus Group

Focus group is a technique used to validate artifacts [104]. It is a direct approach where experts are asked about their professional opinion on the artifacts. However, the participants should have a background on the topic and a good understanding of the artifacts and the context of the problem.

Participant ID	**Code**	**Role**
Participant 017	part.17	Daily Supervisor
Participant 018	part.18	Academic Supervisor
Participant 019	part.19	Operations
Participant 020	part.20	Agile Coach
Participant 021	part.21	Accenture Security
Participant 022	part.22	Accenture Security

Table 3.4: Focus Group Participants List

Unlike interviews, focus groups allow and encourage the discussion between participants to validate the artifact on a holistic level, provided that the session has a clear plan and an appropriate moderation. The final result of the focus group session is to have improved artifacts that are applicable in the problem environment.

In this research, the focus group is used to validate the artifacts created in the case study. A small group of experts were invited for the session to answer three main questions:

1. **Are the models complete?** The goal of this question is to understand if the proposed models have all the components according to the experts.

2. **Are the models correct?** The goal of this question is to understand if all the components are placed in the correct location and have the correct meaning.

3. **What can be improved?** The final question seeks general comments on what can be improved about the models.

After presenting the case study results and explaining the context of the problem, an open discussion took place to answer the aforementioned questions. Participants were given a copy of the models and they were asked to write down their ideas and discuss them. These copies, along with the session's recording, were used to create the final artifacts.

The final findings of this session are reported in section 7. The experts' profile who participated in the focus group session is shown in table 3.4. Two of the research supervisors attended the session. Besides, a couple of interns and new employees have attended the session to observe, they did not engage in the discussion.

Chapter 4

Background

This chapter provides an answer to the first sub-research question. The first part is a literature review of the relevant topic to DevSecOps. The second part reports on the first round of interviews. Both parts give a context on how security is approached in literature and practice.

4.1 Literature Review

4.1.1 Agile and Software Production

Software Development took a sharp turn moving from a Waterfall approach to Agile. This turn allowed developers to deliver software within a short time frame [3]. Clients became more involved in the process and iterations occurred more rapidly. Prior to this milestone change, developers followed traditional project planning approaches to develop software. A project would be broken down to phases: analyzing, designing, implementation, testing and delivery to the client [5] [69]. As successful as this approach can be in other fields, in software production following this waterfall model leads to unsatisfactory results. The core issue with this model is that it is not able to adapt to change. Clients usually come up with new requirements, that is emerging from business, in the middle of a developing phase which can not be included anymore. Additionally, clients were involved only in the start of a project, when requirements are collected, and at the end of the project when the final product is delivered. This limited interaction led to wasting resources on developing software that is not needed. Agile was introduced to change the whole process by allowing iterations, involving clients and shortening the life cycle of software development to allow clients see through what is coming and make iterations where needed [7].

Once the Agile approach was proved useful, other approaches were created on top to be more case specific. Scrum and Kanban are examples of approaches that are based on Agile [40]. Although these two examples are different in the details of the daily processes of software development they both hold on the values that Agile provides. Both approaches are widely adopted in the market and both have certification tracks. Lean![29] is yet another concept that holds on to Agile values [100] and mainly focuses on team collaboration, clients and quick feedback. These examples, and many more, show how Agile changed the narrative of software development as more and more companies apply Agile or a type of Agile.

4.1.2 DevOps

The early implementations of Agile were focused on improving the efficiency of the development phase. These implementations introduced processes that facilitate the development of a desired and working software in a reasonable period of time. However those early implementations, the task of the developer ends once a running code is committed. The following steps are handled by another team, namely: operations, who is responsible for making this committed code ready to use for the clients [11]. While the development team seeks to change rapidly, the operations team wants to create a stable environment for clients [59]. These two different goals were counterproductive. Miscommunication and misalignment between these two teams result in longer periods of delay before deployment [31]. Enabling faster cycles meant breaking the wall between these two teams to empower collaboration from the start of the planning until it gets to users. DevOps is introduced, on top of Agile, to solve the issue of delivering value to clients faster [31].

DevOps, which is a word coming from **Dev**elopment and **Op**erations, is yet another extension of Agile [64]. It is an approach that emerged from the need of breaking the silos between the Development team and the Operations team [32] [59] [91]. DevOps' key enabler is the software delivery model. In a traditional software development, software is delivered to clients based on periodic releases. The release includes one or several updates to a running software which might include new features, tools or fixing of bugs. All of these updates are installed in one released package from development. This model is known as Software as a Product (SaaP) [77] . In the case of web applications and cloud, the delivery method is different. All new updates and features can be directly deployed in the web application server or the cloud and users can immediately use these updates. This delivery model, which is called Software as a Service (SaaS), changed the release planning phase. Updates are no longer bundled by type or importance rather than deploying every feature as soon as it is ready for push.

In literature, there are several definitions of DevOps ranging from very general to very specific. Jabbari et al. [64] conducted a literature review about DevOps' definition and practices where they identified eight components from different definitions, 1) Development and operations, 2) Communication, collaboration and team working, 3) Bridging the gap, 4) Development method, 5) Software delivery, 6) Automated deployment 7) Continuous integration, 8) Quality assurance. Using these components, they came up with a more holistic definition. They defined DevOps as

> "*A development methodology aimed at bridging the gap between Development and Operations, emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices.*" [64].

The definition emphasizes the collaboration between teams continuous integration, and automation. First, team collaboration refers to how both teams work together in continuous cycles of feedback to help and correct each other [64] [91]. This collaboration creates an environment where *everyone* is responsible for the code and delivering value to the client [51]. Secondly, continuous integration means how software can move from a development phase to an operation phase continuously and keep pushing new updates to operations once they are ready [73]. Finally, to increase speed of processes, automation is another driver. DevOps support automation wherever applicable [59].

4.1.3 DevOps Practices

1. **Collaboration:** DevOps brings two different teams closer [91] [101] [52]. Developers not only write the code, but also process feedback from operations about issues within the operation team which need adjustment at the development stages. Operators on the other hand provide input

for developers and both teams become responsible for delivering a working code to clients [40]. Tools, like JIRA[1], Slack[2] and Trello[3], do exist to allow an effective collaboration between teams [11]. Yet, the main issue is the change of culture [71] [40]. Organization's culture in its broader outline is defined by how an organization approaches its goals. Hofstede [58] has identified the dimensions of organization's culture. One of these dimensions was the Individualism and Collectivism. Individualism refers to how employees in organizations are more focused on personal achievements rather than thinking as a team (Collectivism). DevOps is guided by a cultural shift to collaboration [21] where information sharing is the responsibility of both teams [71].

2. **Automation:** To achieve shorter cycles from development to operation, minimizing the manual work is essential [71]. Automation in DevOps appears in most stages: build automation, test automation [31], integration automation [101], configuration automation [31] and deployment automation [64]. These concepts collectively are referred to as the delivery pipeline [101]. In a pipeline, an artifact goes through the different stages, namely: build, test, integrate and deployment, hence automatically reducing time, effort and manual errors. In case a test was marked unsuccessful, the process is aborted [31]. Many tools exist to support automation, in fact, each of the mentioned areas has a full set of tools, such as Jenkins[4], puppet[5] and Selenium[6] to name a few. Therefore, selecting a proper "tool chain" is a critical decision [11]. Tools are classified by the tasks they are performing, i.e. testing, build, configuration, or by their source, i.e. open source or commercial tools. They can also be classified by scale where some tools are good for small organization, other tools work best on large-scale enterprise setting.

3. **Continuity:** Automation implied perpetuation of process throughout the life cycle of an application. Continuous Integration and Continuous Deployment (CI/CD) are fundamental concepts in DevOps. Powered by automation tools, the process of integrating the code and deploying

[1] https://atlassian.com/software/jira

[2] https://slack.com

[3] https://trello.com/

[4] https://jenkins.io/

[5] https://puppet.com/

[6] https://docs.seleniumhq.org/

it becomes efficient [84]. Continuous Integration is the process of connecting the different components of a software on top of each other into one working software. The components can have different sources or produced by different teams [65]. Continuous Deployment, on the other hand, is the process of delivering the final good code to the production environment where it is ready for use [78]. The concept of continuous* (pronounced: Continuous Star) emerged to extend over the need for instant change beyond CI/CD. Continuous* includes not only build, testing, integration, deployment but also goes further to cover other DevOps processes such as continuous planning, continuous use, continuous monitoring, continuous feedback and continuous security [40].

4. **Monitoring and Measurement:** Monitoring is part of the operations team tasks. It involves monitoring the running applications and how resources are allocated. Additionally, it contains monitoring the infrastructure and report problems when they arise and address them if possible or notify a team member [71]. Monitoring is a process that tells if a system is doing what it should be doing, or alerts the staff when this is not the case. To be able to determine the Behavior of a system, a concrete measurement model is needed [89]. This model is based on success factors emerging from the business, if these are not well defined, it is hard to determine how successful the implementation of DevOps is and the whole monitoring purpose becomes pointless [40]. The results of monitoring are organized and used in the feedback loop for the next development iteration.

4.1.4 DevOps Stages

DevOps has multiple frameworks. Each organization has its own approach of adoption, chain of tools, principles and policies [32]. In general terms, the DevOps cycle contains two main components: development and operation. At the development stage, the team uses the feedback from the operations stage, along with other sources, as input for their phase. On the other direction, the operation stage takes the code produced by the development stage and deploy it in the production environment. The process takes place in several virtual environments organized in a "*pipeline*". Microservices represent the independent features, bug fixes and quality enhancements to be developed and deployed. The development process, showed in Figure 4.1, starts by

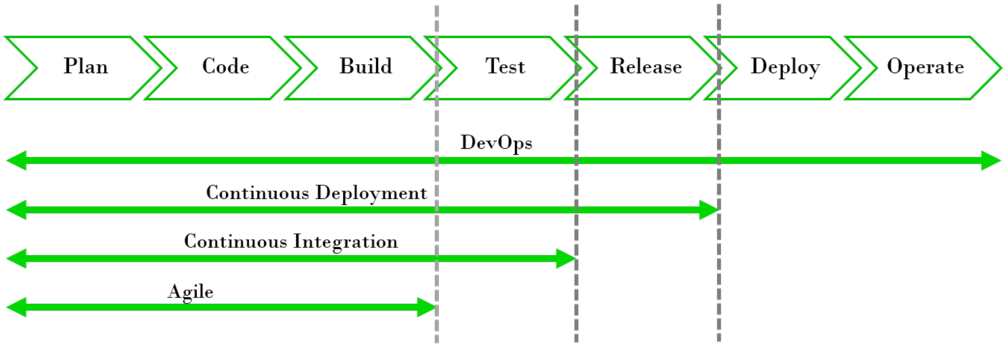


Figure 4.1: DevOps Stages based on Rathod and Surve [83]

planning where the development team prioritizes requirements in the planning stage. Once tasks are selected, resources are assigned and coding starts. In this second stage, the developer follows a predefined style of coding which aligns with what other developers are working on. Before starting a build process, teams peer-review each others code to ensure the quality of the code. The build is the process of compiling the source code to produce a software artifact. If the build has issues, the process is aborted however, if it was successful, it continues to the last step in the development phase. Testing is a process of examining if the produced code is behaving as expected and handles unexpected situations without breaking the software. Testing is a large process that contains functional tests, quality tests, security tests and performance tests [59].

Applying DevOps requires automating development and operations processes. In particular the build, test, release and deployment stages [91]. Once a developer commits a code, the code goes through a pipeline of automated processes. First, the server, i.e. Jenkins server , will pull the committed code and creates a new build for the software. If succeeded, the code proceeds to the testing environment [83]. In this stage, the code is tested against prepared and updated tests for Quality Assurance (QA). Containers, such as Docker[7], can be used to create dynamic testing environment that is easily destroyed after the test [50]. If the code did not pass the tests, the process is aborted and the developer is notified. After passing the tests, the code is ready for release and deployment. The code is moved once more to the production

[7] https://www.docker.com/

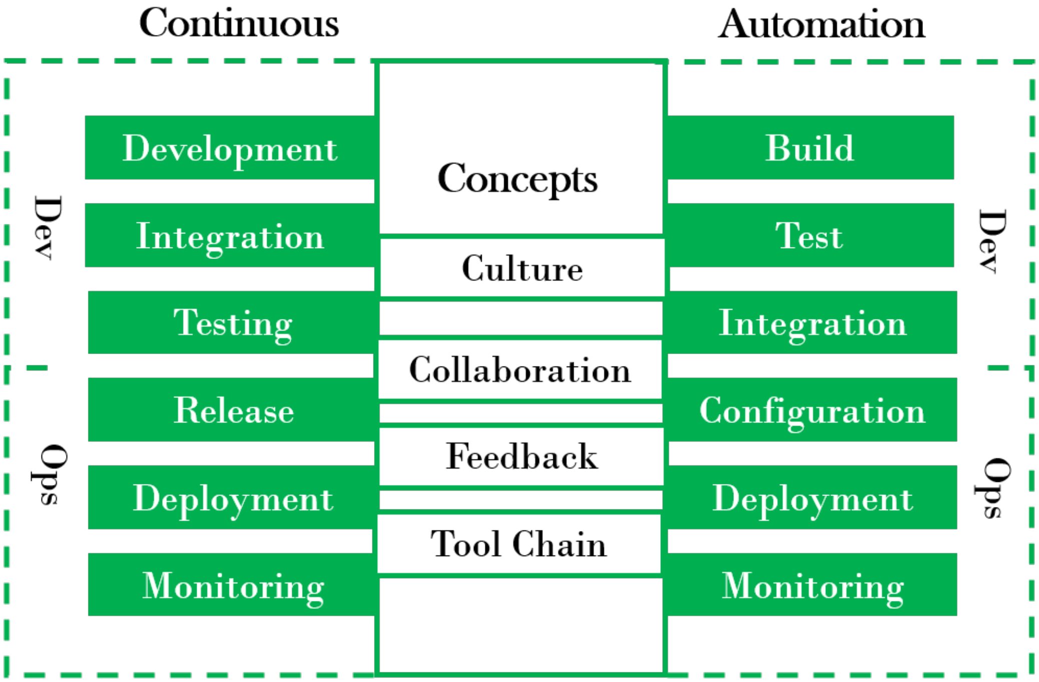


Figure 4.2: DevOps Outline

environment using a delivery model such as blue-green deployment [50]. All of these processes can happen hundreds of time per day. DevOps outline is summarized in Figure 4.2.

4.1.5 Security in Software Production

Simply producing a code that is performing the required task is no longer enough. Security of applications is a critical factor for businesses to select software vendors. Security breaches happen when a code is used unexpectedly to influence security principles: confidentiality , integrity, availability, authentication, authorization and nonrepudiation of software [38] [23]. Such behavior can harm users, businesses or governments. Security principles define the borderlines of security. Table 4.1 concludes how these principles are defined in literature. Integrating security within the application is a challenging, yet required, task.

Starting from the very bottom, security requirements are considered part

Term	**Definition**	**Attack Example**
Confidentiality	is the assurance that information is not disclosed to unauthorized individuals, processes, or devices.	SQL injection
Integrity	is provided when data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed.	Deserialization attack
Availability	guarantees timely, reliable access to data and information services for authorized users.	DDoS attack
Authentication	is a security measure designed to establish the validity of a transmission, message or originator, or a mean for verifying an individual authorization to receive specific categories of information.	SQL injection
Authorization	provides access privileges granted to a user, program, or process.	Session hijacking
Nonrepudiation	is the assurance that none of the partners taking part in a transaction can later deny of having participated.	Repudiation attack

Table 4.1: Software Security Principles [38] [19] [87]

of the non-functional requirements which include performance, usability, robustness and many others. Security requirements are not the same for each application or environment. Once identified, they are planned for development and testing. However, the main observation is that security features are slowing down the delivery process [73]. In traditional software development, security tests are performed after the code is developed and tested. A tester would eventually hand in a list of changes and bugs that must be fixed before a release is approved [15]. Considering a DevOps approach where software is rapidly changing, security hinders the overall process.

Vulnerabilities are defined as system characteristics that allow for a class specific security breach [38]. In other words, they are weak points in the system that enable attackers to compromise the security of an application. These vulnerabilities can also exist in open source libraries [72] which developers reuse to build their own applications. Web applications and cloud-based applications are more accessible to attackers than closed secret systems. Other emerging technologies such as payments systems, IoT and smart phones in health introduce serious risks [20]. OWASP[8] has identified the top ten most potential threats in 2017 [1]. This open source project aims to educate developers, product owners and web application community about the baseline security quality to take in consideration when developing an application. The following compiles the list of 2017.

1. **Injection:** is one of the most common types of attacks. The nature of this attack can impact many software technologies such as SQL, NoSQL and XML. The attack is based on sending untrusted commands through a string parameter [17]. This data is then executed at the server side resulting in unauthorized access, disclosure of data or denial of service [80]. According to the OWASP metric [1], this exploit can be easily launched, although more complex attacks can be created, and the technical impact can be severe. Preventing this attack includes adding extra layer of coding that examines the executed input and output. Yet, the main challenge is how to distinguish attacks from legitimate queries. For this, machine learning techniques can be used to create classifiers that can identify the nature of the executed commands [99].

2. **Broken Authentication:** is an attack that aims to gain access to a system through passwords, keys or session tokens. In particular, attackers are using combinations of common user names and passwords or by guessing them to gain access and compromise the system. This attack is heavily based on password and key management. For example, systems that create a predefined password for users are in high risk. Additionally, session tokens are exploited if not configured properly. Attackers can reuse authenticated tokens to gain access [54]. Covering this weakness requires installing password mechanism, that checks for weak passwords, and using multi-factor authentication. It also includes reviewing the session management flows and use of Single Sign-On tokens (SSO) that expire after one use [1].

[8] https://www.owasp.org/

3. **Sensitive Data Exposure:** is targeted towards acquiring data from servers (data at rest), in transit or at the client's side, i.e through browser. Attacks aim to get sensitive data such as credit card number or personal identifiable information (PII) which is used to commit fraud or identity theft, leaving users with severe impact [26]. Companies can also be held accountable for not adhering to local personal regulations like General Data Protection Regulation (GDPR)[9]. On the application layer, this attack happens when text is not encrypted when transmitted using HTTP, SMTP or FTP protocols or when old, weak encryption algorithms are used. To improve an application against these attacks, PII data must be classified and encrypted in all states. Besides, encryption algorithms need to be updated, enhanced and enforced by the application [1].

4. **XML External Entities (XXE):** is an XML attack. In fact, it is a feature used in older versions of web applications to allow XML parser to execute external code. However, like other injection based attacks, when untrusted code is parsed, it results in a denial of service attack. One example is the Billion Laughs attack [70] where a small code can result in a DoS attack. Eliminating this vulnerability starts with disabling External Entity parser on the web application if possible and implementing a safe server-side validation and filtering of input.

5. **Broken Access Control:** is an attack aimed to get privileges in the system. Either a user performing administrator-level functions or a visitor performing users' tasks. The access controls are not easily automatically detected. Some tests, like the static code analysis, can indicate if the system has access controls, yet, it is not possible to know if they are functioning properly. Attackers would bypass controls by modifying URLs or use unauthorized API access. Such a breach can result in data loss, disclosure or alteration. To defend against these attacks, critical pages must not be accessible without authentication and disabling the web server directory listing. Besides, on a defensive aspect, failed access controls are to be logged and admins are notified [1].

6. **Security Misconfiguration:** is yet another threat that can be prevented with proper security and testing [34]. Applications come with

[9] https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=IMM14202GBEN

predefined default parameters such as user names and password, ports, unpatched applications and services. The unused parameters are searched by attackers to take over a system. These default values can be on the infrastructure level such as server OS, containers, web server, i.e. apache, database, or the application itself [93]. A developer needs to identify the services needed to perform the task of the application and switch-off any unused service. Additionally, regular updates and security patches are to be taken seriously to ensure the highest level of security.

7. **Cross-Site Scripting (XSS):** is one of the most widespread attacks according to OWASP. The idea is to allow an attacker to execute a script on the user's browser allowing him to steal cookie or credentials of the user. The attack starts by posting a malicious script to a vulnerable web application. Once a user accesses the web application, he will get a response from the web application including the malicious script [86]. The code will execute on the user's machine and will initiate a connection with the attacker to send the information stored on the browser. Preventing this attack has two layers. First, at the server-side, applications need to isolate untrusted posted code from the active application content. Second, at the user-side, browsers are improved to escape executing malicious code and escape HTTP requests [1].

8. **Insecure Deserialization:** is a new item on the top ten list. Serialization refers to the process of decomposing an object to byte stream in order to be sent over a link for processing. In the reversing operation, deserialization, the object is created back from its stream of bytes. If there was no validation for the objects, an attacker can change the values of the object. For example, in case of a cookie, an attacker can change the role from (user) to (admin) bypassing the access control. This exploit is difficult to create and securing against this threat includes limiting the use of serialization to basic data types and include validation of objects before and after serialization such as using signatures [1].

9. **Using Components with Known Vulnerabilities:** is a threat caused by using third party software or libraries. It is partially related to threat number six, security misconfiguration. Yet, in this threat the

software itself is vulnerable. These known vulnerabilities make it easier for attackers to start a specific type of attacks, whilst knowing the weaknesses makes it easier for security specialists to cover and guard them [13]. Preventing these attacks start by having a list with all components used and their versions. Updating components is very essential to cover security issues and protect the application.

10. **Insufficient Logging and Monitoring:** is the last item on the list. It refers to two different, yet connected, activities. "*Logging*" is the process of creating an entry whenever an interesting event has happened such as a failed login, while "*monitoring*" is the process of analyzing these logs and making judgments based on them. The core issue here is when logging is poorly implemented in a way that allows attackers to have access to these logs or when the alerting thresholds are not properly configured. In both of these cases, a breach or malicious activity will not be recognized. To ensure the security of the application, logging and monitoring must be introduced and managed throughout the life cycle of an application [1].

The OWASP list provides an overall view of the currently faced threats, however it is not a comprehensive list. This list can be considered as a security baseline against most common threats and extra measures need to be introduced on top to defend against more complex and targeted attacks.

4.1.6 Security Testing

Testing is a phase that follows development to examine the developed code. It consists of running multiple manual or automated tests to measure the over all quality of the code [38]. This broad term refers to different test areas including functional testing and non-functional testing. In functional testing, the testers check if the code is performing as expected and produces the correct results. On the other hand, non-functional testing focuses more, among many others, on performance, robustness and security [23] . Each has its own methods and set of tools. Besides, not all types of tests are equally important for *all* software. Some software can value performance more while others might value security more.

Testing is categorized into two main branches, white-box testing and black-box testing. Although these categories are valid for other tests, the focus here is on security testing. In a white-box test, the tester, who is playing the role of an attacker, has knowledge about the system and how it is working and possibly the source code. He then uses this advantage to create the security tests based on that knowledge with the goal of compromising the system. The black-box testing, on the other hand, is from an outsider prospective. The tester, or attacker, would have very limited knowledge about how the system is handling processes. Therefore, the attacker must be creative to break into the system [9] [23] [105]. These two broad categories contain within many types of security tests that ensure that all the vulnerabilities are secured.

Penetration testing is a type of tests aimed to stress the security of a running software [17]. The goal of this test is to try exploit a running code and find possible vulnerabilities and then report them back to the developers to be patched [68]. The test simulates, in a safe environment, what a hacker might do in order to exploit the software [94]. There are tools used to generate the attacks automatically such as Zed Attach Proxy (ZAP)[10] which is developed by OWASP [43].

Penetration testing approaches such as fuzz testing and combinatorial testing are also used in the functional testing field. Fuzz testing (Fuzzing) is a randomly generated test aimed to find code vulnerabilities. The input parameters are mutated or generated to test how an application will respond to unexpected or invalid types of input parameter [43]. Fuzzing tools are based on intelligent model-based test [94] [105]. Combinatorial testing is yet another approach focused on test cases. The tested software, also called system under test (SUT), is given "N" number of cases that cover the tests that might result in unhandled exceptions. This test has evolved to cover the issue of exponential grow of test combinations [105].

4.1.7 DevSecOps

DevOps is indeed well adopted in software development, yet the question of how security is handled is a challenge. While DevOps is going for more speed and agility, adding security tests and practices hinders the rapid processes of DevOps [73]. DevSecOps, which is also called in some literature Secure DevOps, is an initiative that aims to adapt security practices to fit in the DevOps processes [77]. Gartner defines DevSecOps as

[10] https://www.zaproxy.org/

> "*The integration of security into emerging agile IT and DevOps development as seamlessly and as transparently as possible, ideally without reducing the agility or speed of developers or requiring to leave their development tool-chain environment.*" [45].

The definition has three main components 1) **adapting security practices:** indicates that security practices must be included in the DevOps framework to ensure the security of the software and safety of the information processed [15]. 2) **maintaining DevOps teams speed and agility:** this concept refers to the need of refining security practices to fit-in DevOps and not the other way around. Security practices need to adhere to the agility of DevOps [46]. 3) **maintaining the same development tool-chain:** indicates that security tool should be integrated into the development environment and developers can use them without the supervision of a security expert[44].

DevSecOps is an extension of DevOps. The concept of DevOps emerged to solve the issue between two teams Development and Operations. On the same way, DevSecOps is aiming to break silos between the DevOps team and the Security team in all stages of the project and not only in the testing stage [15]. This is identified by "*shifting security left*" [100] [73] [96]. Additionally, DevSecOps promotes the idea of software security is the responsibility of everyone and all members need to think about security at all times. Developers in the development stage must be educated about the security threats and they must take these threats into account when developing code [46]. The same applies to the operation team who must pay attention to security threats such as misconfiguration and predefined passwords. It requires both teams to synchronize efforts to improve the level of security.

The principles of DevSecOps are derived from DevOps principles. First of all, DevSecOps changes the culture of all teams who are required to collaborate and plan security in the planning stage [77] [75]. Automation of security tests is as important as automation of functional tests in DevOps to keep up with the over all speed [77]. Test should not require security experts to perform [44]. Additionally, security events must be recorded and monitored by security teams [77]. These measurements are adjusted for different audience and for different purposes [20]. Finally, the sharing environment in a DevSecOps team is encouraged. The "s*ecurity champion*" model can be used to stimulate sharing. The model implies training a developer on security practices and then this developer becomes a security reference (champion) for the whole development team [15].

On a practical level, DevSecOps promotes security-by-design concepts [25]. Security starts by creating a security policy that is considered as a reference point for software security. This policy is polished and improved along course of the DevSecOps cycle. Authentication, data protection, logging and monitoring, vulnerability and patch management are few examples that are enforced by DevSecOps. Application's security starts by planning the correct security measures and proper security testing such as threat modeling [44]. These tests are not carried out only before deployment, but also during the operation phase in what is called "*continuous testing*" [77]. Further, DevSecOps requires additional tools that handle security aspects, in particular, tools that facilitate the automation of security test and monitoring of security logs [75].

4.2 Interviews

The following section describes how DevOps is defined by practitioners and where security is taking place in the process. Before the interviews, two assumptions were made based on the literature review. First, each interviewee will have a different definition for DevOps as pointed out by Elberzhager et al. [32]. Second, security is not clearly defined within the team or it is the responsibility of another team.

4.2.1 DevOps in Practice

In practice, DevOps is defined differently by the practitioners, while some focused on the process of speeding delivery, others talked about culture and collaboration. All interviewees mentioned that the whole idea is two bring the development and operation teams together. These attributes are mentioned in the definition of Jabbari et al. [64]. Further, the stages of DevOps in practice are similar to the framework proposed by by Elberzhager et al. [32].

> "*I think for me personally, how I see DevOps and what it means is a quick way to make a change along with the desired quality ...*" [int.01]

> "*It is like a practice of developing and operating at the same time. It requires 1) understanding the application requirements.*

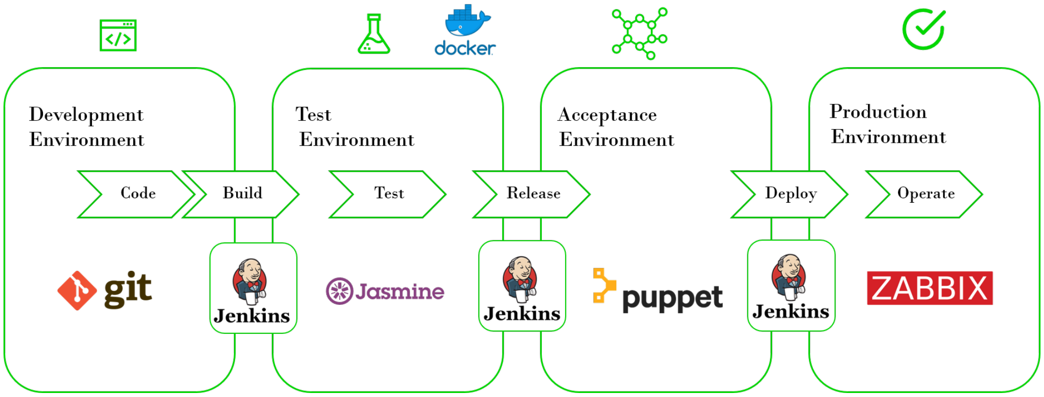


Figure 4.3: DevOps in Practice

> *2) Characteristics. This includes culture, automation and monitoring for the whole life cycle of the application. It works for small releases, upgrades, bug fixes and new features. DevOps come hand in hand with CI/CD (continuous integration / continuous delivery).*" [int.02]

An interesting point of view is the T-shaped professional. The interviewee demonstrated that DevOps team member should have two types of expertise forming the letter T. A member, a developer for example, should have variety of knowledge in development, testing, server management and many others and that is the horizontal view. Yet, this developer should have deep knowledge in his field forming vertical level of expertise.

Typically, a project would have 4 main environments as shown in Figure 4.3. 1) Development Environment is where all the coding and testing take place. 2) Testing Environment is used to conduct all types of tests. 3) Deployment Environment, which is also called Acceptance Environment, is where all the working and approved code is integrated and stored. 4) Production Environment is the environment that is accessible to users which is also called production environment. Additionally, all interviewees pointed that they worked with a different level of maturity of DevOps and not all steps in the pipeline were fully automated because of the nature of the software or the bureaucracy of procedures.

4.2.2 Security in Practice

Security was a task handled by the security team or the integration team. All interviewees articulated that security was not the responsibility of the DevOps team. A security test is usually conducted when the increment is to be moved from the testing environment to the acceptance environment. Only in that stage, the security team is involved with the process. This results in two main issues. 1) The security team is involved when the code is already built and tested. If the testing fails, pushing the increment will be definitely delayed. 2) Being responsible for the increment is no longer valid, because the team is not responsible for the security.

> "*The way we think about security is something that costs time, something which we remember after we have defined our requirements. After we define the planning we realize that we need to think about security. It did not feel at that time as part of our responsibility but that there is other [team] which is [assigned] and accountable for security and that we need to have their approval, their go [and] signs for everything, this is how it felt... it is their test, it is their thing and not our thing.*" [int.03]

On one of the bigger projects, a company had a big release that will improve the user experience of the application. This was an important release that the higher management was involved on daily stand-ups to push the efforts of creating a successful release. The project had many DevOps teams working on a multi-layered application. There were many technical issues and it was a challenge to finally and proudly submit the release 3 weeks before the deadline, which was enough time to conduct all security tests. The release failed terribly to meet GDPR requirements on different levels and the release had to be delayed for 3 extra weeks and the release had to be decided on the highest level. "*...it was painful.*" [int.03].

Including security in the earlier stages of planning an increment is key. The interviewees stated that involving the security team in the planning stages can reduce the probability that an increment fails because of security related issues. Interviewee [int.04] stated that security team, with its knowledge and tools, should be included in DevOps. On the other hand security is handled by the "*integration team*" which is part of the operation team. Yet the only note is that the integration team is involved very late in the process.

> "*Yes, we have a specialized team called integration team, they take care of security and other things like deployment to put the code into production we have a pipeline in place, but we don't control deployment to production.*" [int.04]

Security on DevOps does exist on some applications, depending on how critical the application is. However, in all cases they are involved very late in the process and that results in failing increments. The concept of security is exclusive for the security team who is eventually responsible and accountable in case of a security breach.

4.3 DevSecOps in Theory and Practice

An over look into DevSecOps in theory shows that it is a new topic with limited theory around it. The main papers shaped the main practices of DevSecOps. However, detailed view of the topic is still missing. On a practical level, experts see the urge and need for such a concept, yet, there is no clear view on what is it about or how to approach it in day to day activities. The key points projected in this review that security needs to be involved in the whole process of creating software, starting from the planning stage, then testing and in production as security seen as a continuous process. Based on that, different level of security checkpoints will take place along the pipeline of creating software. Figure 4.4 summarizes how security is understood in theory and in practice.

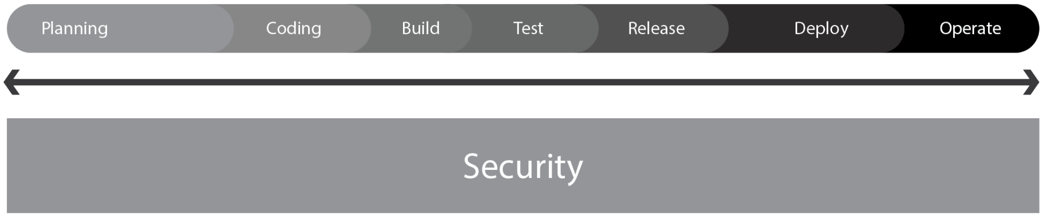


Figure 4.4: DevSecOps in Literature and Practice

Chapter 5

Impact Areas of DevSecOps

This chapter describes the first artifact. The DevSecOps impact model aims to illustrate the main areas of change when adding security to DevOps. These areas are inspired by DevOps practices in real-world organizations. Figure 5.1 shows these areas and the components of each area. This is not the final artifact, a validated model is presented in section 7.1.

5.1 Towards The Impact Area Model

Figure 5.1 represents the impact area model as proposed by interviewees and literature. The interviewees proposed these areas based on their experiences in transformation to DevOps. Triangulation [104] is used to collect *all* the areas from both interviews and literature. To make these areas easier to read, they were grouped into four main groups namely: People, Tools, Values and Processes. Part of these areas, such as measurement and automation, were discussed in the literature review section. The interview protocol included these items explicitly. However, other areas such as ownership and accountability were raised only during the interviews.

5.2 People

This concept includes different stakeholders working in the DevOps including the management, DevOps team and end users. Each stakeholder will have a role in enhancing and ensuring a certain level of security. Security does not start or end with one party, rather the entire group of people is involved in

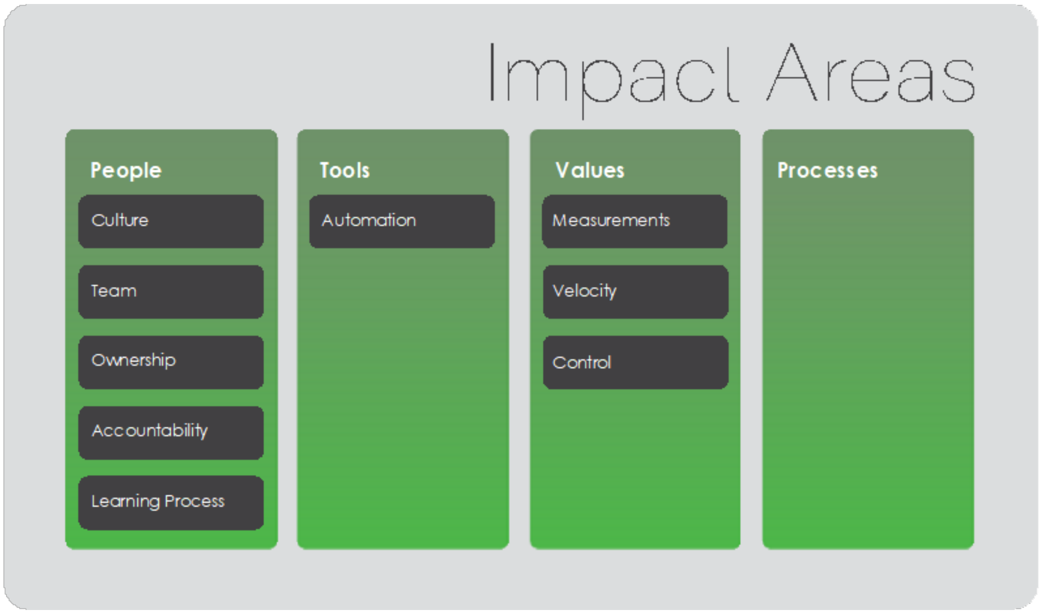


Figure 5.1: Impact Areas of DevSecOps

the process. Having a DevSecOps mindset ensures that security has a very high priority as the people realize the consequences of not applying security in place.

5.2.1 Culture

DevOps has extended the agile culture to allow the team to work dynamically and autonomously. Adding security to DevOps team can have two different views. In one hand, changing the mindset of teams to improve security can be seen as a culture change. Teams are required to approach security on many levels during the different stages of DevOps. On the other hand, DevSecOps preserves and embraces the culture of agility and short delivery of software, thus viewing it as the same culture.

> *"It should change the culture, it should change the mindset of the team. It should change how the team think about functionality and availability and to care also about how secure this is. I think it should."* [int.09]

> "*In general, what we see is that security is seen as something separate. (Security is) seen as a part of the IT department, auditors and checklists. But if it was part of the routine, part of the daily testing cycle, people should not notice that they are doing security even though they are.*" [int.10]

DevSecOps culture is an extension of DevOps culture, it has the same values, yet it has its own mindset where security comes at the core of building software.

5.2.2 Team

When focusing on security, new skills are required from all the members in the team. It is not obligatory that all team members have a deep knowledge of security, however, they are required to have a certain level of understanding of security. T-shaped professional, as mentioned, refers to how skilled a team member can be. Looking at the letter "T", on the horizontal level, a team member should have some understanding on all the stages of DevOps. On the vertical level, this same team member must have a very deep knowledge in his own area of expertise. For example, for a tester, he should have a good understanding of planning requirements, developing code, deploying code to different environment. At the same time, this person must have a deep knowledge on testing, his main core expertise [int.03] [int.15].

In this context, adding security to DevOps requires the team to add a new specialty to their T-shaped profile. All team members will have at minimum a basic understanding of security and threats that can risk the application. This concept, the T-shaped profession, also implies that there must be a person, who is part of the team and involved in the whole process of DevOps, that can handle security. The security specialist can work with multiple security teams at once, however, this depends on the application and the level of complexity it has. This change in the team composition requires re-integration of the team members so that no silos are created within the DevSecOps team. Although a security member is included in the team, security as a discipline should remain as an organizational structure. The focus of this department is to provide a higher level of support for security specialists within their teams.

5.2.3 Ownership

> "*In some companies, teams own small parts of the software which are called microservices and everything is in their own hands, that is the ownership.*" [int.07]

The idea of ownership in DevOps indicates that a team is the owner of an increment. In a microservice architecture, a team is the owner of the services created by the team. This includes building and operating the service as well as solving all future issues this service can generate. In practice, the team is the owner of all parts related to their microservices, except for security which is handled by the security department. In DevSecOps, ownership is extended to include security. The whole team becomes owner to all the increments they are producing end-to-end. Security issues will be handled within the team just like other issues [int.05] [int.07].

5.2.4 Accountability

Accountability goes hand in hand with ownership. Now that the team members are working independently end-to-end and becoming full owners of an increment, they also become accountable to their work. In case of a faulty software with security issues, the whole team is to take responsibility for their work and the whole team has to work together in solving the issues [int.09].

5.2.5 Learning Process

People will also need time to learn. Security is just another skill that all team members need to learn and apply in their domain of work, thus adding an extra skill to the T-shaped profile. Security is a general term that includes many concepts and areas. The members of the DevOps team will have different areas of interest that best help them apply this knowledge in their work. Additionally, training the team to develop secure code looks very expensive, however, the cost of implementing security afterwards is higher [15] [int.08] [int.10].

5.3 Tools

In a full end-to-end DevOps pipeline, many tools are used in each stage. Tools are key in ensuring consistency, agility and automation. Security on the other hand has its own set of tools that allows security experts to examine the code and discover possible vulnerabilities. These tools are to be adopted by the DevOps pipeline to facilitate the purpose of end-to-end DevOps.

5.3.1 Automation

Automation means that computers are used to execute tasks. What makes computers powerful in automation is that they can execute the same tasks multiple times in a very short time and in a consistent way. For example, a tester might run a manual test perfectly the first time. However, when performing the same test for the tenth time, it will not be as accurate. In security this concept is amplified as similar tests will be executed during the process of developing an increment. In a DevSecOps setting, security tools that allow automation have a critical role in ensuring security while preserving the team's agility [int.08] [int.11].

> "*I always say, people should think, machines should execute.*" [int.05]

5.4 Values

5.4.1 Measurement

DevOps allowed teams to measure their improvements in a different way. At each maturity level, the team is looking into a different set of (Key Performance Indicators) KPIs that indicates the team's capabilities. For example, the number of builds created, the number of integrations per year and number of builds failing in a test environment. These measurements help the team realize their strengths and weaknesses. When a team moves to a higher level of maturity in DevOps, new KPIs are set in place [int.06] [int.14].

> "***Velocity*** *is a common agile KPI, it says: only I commit something, and how good I am in estimating what I can commit. Then*

> *the next level, you can start measure your technical depth. But you should grow into that, you should not use them while learning agile. You need to create a journey.*" [int.14]

As an extension to DevOps, DevSecOps must preserve and brace these measurements, at the same time introduce new KPIs that describe the security aspect of a software. As a start, teams need to apply security KPIs that reflects their level of maturity in security. It is also crucial that these measurements are thought of at the start of planning an increment, because the measurement tools will be built along as well. When KPIs are an afterthought, it will take the team extra time to build or find the correct tool to perform the monitoring afterwards. Taking testing as an example, a starting point for measuring security tests is the percentage of automated tests compared to manual tests. The percentage of automated security tests are expected to increase as the team matures. In a higher level of maturity, the focus will shift into how the application can recover in case of an incident or how it should act while it is being attacked.

5.4.2 Velocity

Velocity refers to how fast a team can deliver. Whether it is a new feature, a bug fix or improving current code, velocity is key in DevOps teams. Bringing the development and operations has increased velocity and the initial thoughts that adding security will increase velocity even more. Although the initial thoughts of adding security to the team will result in extra steps that costs the team extra time to finish a task, including security in the whole DevSecOps process eliminates the need of consulting a security team before going to production, less rollbacks and less technical debt. Thus increasing the overall velocity of the team. At the beginning of applying DevSecOps, teams are expected to lose a portion of their velocity as they have to go through a learning curve. However, once that is accomplished, the velocity will increase [int.14].

> "*... if we embed the security people in the team, we deliver the product together and security becomes part of the delivery already, so we have no external dependencies. We do not have to go to another department to check if something is possible or not since security ownership is part of the product delivery. I think that will speed (up) the process. It will remove the dependencies.*" [int.05]

5.4.3 Control

Control is related to ownership and accountability. Control in its broader sense refers to how the team is managing a DevOps increment end-to-end, from the start of the idea on a sprint planning until it is operational in a production environment. In practice, a team would be in control of the whole increment end-to-end. Except for security which is always outsourced to the security department. This blind spot gives a DevOps team less control on their own product and create dependencies which in turn impacts the velocity of the team. When security is insourced within the team, dependencies are removed and more control is given to the team [int.16].

5.5 Processes

In the DevOps pipeline showed in Figure 4.1, there are stages that describe roughly how DevOps is approached. This line of stages can be tailored per organization or per application following the context and the technology used in building an application. In DevSecOps, these stages remain unchanged and the main changes are happening within each stage. These concepts are detailed in the following chapter which is forming the DevSecOps framework.

Chapter 6

DevSecOps Initial Framework

This chapter describes the second artifact. DevSecOps framework is a representation of both DevOps processes and security practices. The framework indicates what security measures are to be taken in each stage. Not all of these practices are required for each increment and that is going to be elaborated on in section 6.2.2. The framework is composed of three main areas: the initial baseline security, the DevOps environments and stages, and finally the infrastructure level of DevOps across environments. All of these components are explained in the following sections. This is not the final artifact, a validated model is presented in section 7.2.

6.1 Initial Baseline Security

Each application has its own context and its own level of security. Applications that are used for managing appointments will have a different level of security when compared to applications that handle financial transactions. An initial baseline of security is a set of rules, principles, and practices that an organization develops and improves in response to the context and importance of the application [47]. This initial baseline of security ensures that an application has the minimum required security for each increment. One example of this initial baseline security is standards and regulations. For an application, being compliant to standards and regulations can ensure a level of security, however, it does not mean the application has an appropriate level of security [96].

For applications that have multiple DevOps teams, having this baseline

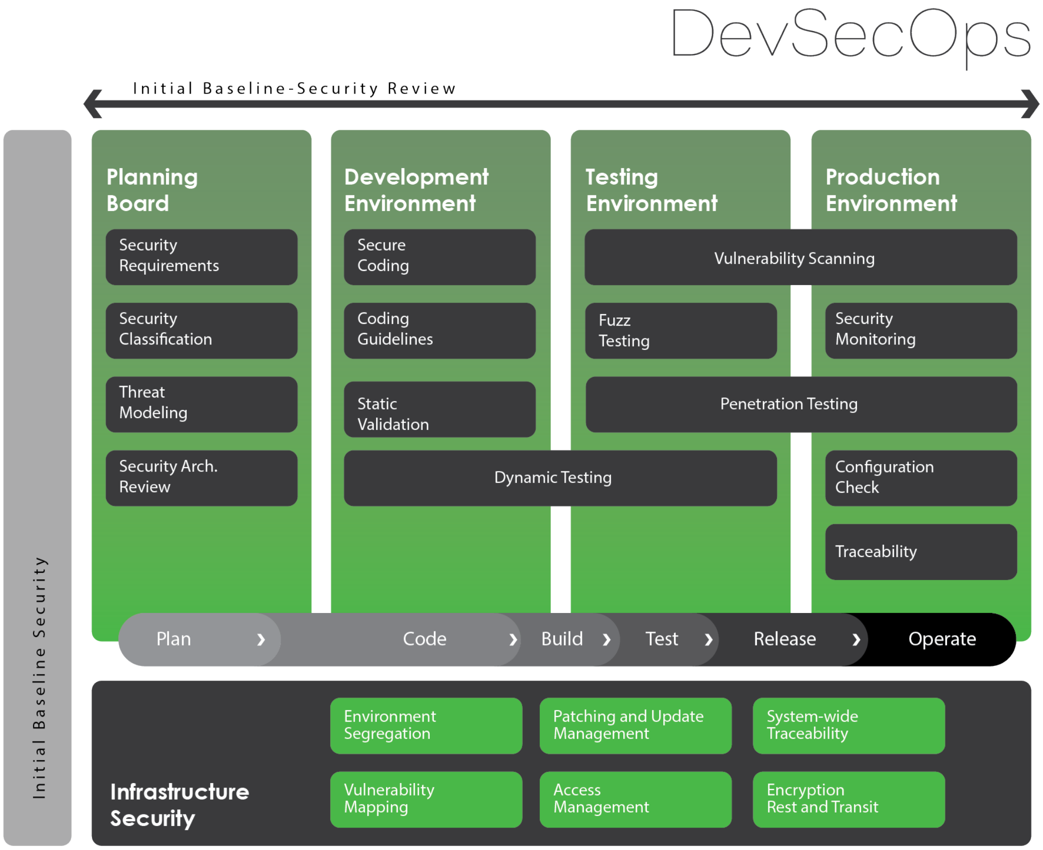


Figure 6.1: DevSecOps Initial Framework

policy is essential. When teams mature in DevOps, they start tailoring DevOps to their own and their application's needs. They become professional in creating their own processes, tools and checks. The risk is when each team works in their own way to secure the application with no reference point that ensures the unity of security levels across all DevOps teams. In this context, having an initial baseline security means that all DevOps teams are expected to have the same minimum security.

Security changes over time and teams need to adapt to change. When new technologies emerge that can extend the teams capabilities, initial security baseline must be updated to include these new technologies. One example is the use of containers. Containerization is a new technology that empowers a DevOps team by allowing them to create and destroy environments. These environments come in with all required applications making it very convenient for testing or operation. Yet, there are new risks that come along and that might not be covered in the initial baseline. One of these risks is kernel exploits where a container can take all host resources resulting in a break down of hosts. Updating the initial baseline security allows the team to use these new technologies and know their risk and how to approach it. Updating the initial baseline security is achieved by a review that happens throughout the pipeline stages where the team reviews if security changes are to be improved. Changes will also impact other DevOps teams [int.09] [int.12].

> "*You should have a baseline for all of your applications, then you scale up based on how critical the application is based on the CIA (Confidentiality, Integrity, Availability) model.*" [int.12]

6.2 Planning Board

Before starting with an increment, a planning session will take place. In Scrum[1], it will be the sprint planning where the team decides which new features are to be included in the next sprint. The product owner gives enough room for the team to select the next set of items from the product's backlog. The selected items, whether they are new features or bug fixes, they will bring a level of security risk that needs to be addressed within the planning session and discussed within the team before proceeding. To approach this purpose, a set of mandatory actions must take place in the planning stage and these are the Security Requirements and Security Classification. The other two items, namely: Threat Modeling and Security Architecture Review can be optional, based on the results of the security classification. The following is a detailed description of these items.

6.2.1 Security Requirements

> "*Developers who understand the security requirements are rare and expensive, normal developers will look at security requirements as world evil and try to skip that somehow.*" [int.13]

Security Requirements are non-functional requirements that captures and mitigates security threats in software. They are created to preserve the

[1] https://www.scrum.org/resources/what-is-scrum

security concepts, namely: confidentiality, integrity, availability, authentication, authorization and non-repudiation [39], introduced in section 4.1.5. These requirements are elicited and prioritized along the process of identifying the functional requirements. Ramachandran [82] has described existing techniques for specifying security requirements such as Attack Tree [2] and Attack Pattern [3]. There are also newer methods that model threats such as the method described by El-Hadary and El-Kassas [30] where they require modeling the system and identifying the threat and possible vulnerabilities. Based on that, security requirements are built and connected to the system's functional requirements as constraints describing how the system will handle the identified threats [int.06] [int.08] [int.11] [int.13].

Just like functional requirements, security requirements need to be explicit, complete, concise, understandable, and unambiguous [36].

6.2.2 Security Classification

Specifying the security requirements leads to a better understanding of the risk introduced by a new feature. The requirements will explicitly state what process flows, data or services that will be changed or altered. This leads to the second major step which is classifying the security level of the feature. Putting security labels on features determines the security controls and the actions to be taken in the following DevOps stages. The urge of classifying the security risk comes from the fact that not all features or changes will introduce high risk, therefore making it inefficient to conduct all security tests and perform all security controls to all features or changes [int.09].

Although an organization can have its own way of classifying risk, OWASP has a classification model, Figure 6.2, that takes into account two main variables, impact and likelihood. *Impact* refers to how severe an attack can be and *likelihood* refers to how likely an attack can happen, that often refers to how easy launching an attack is or how wide spread the attack is. At the intersection of these two variables is the risk of a feature. An abstract example of using this model is a new feature with an API that handles client's authentication to view booking information, on the likelihood axis, this API can produce SQL injection attacks which are very likely to happen. On the impact axis, a successful attacker might gain access, however, they can only

[2] https://www.schneier.com/academic/archives/1999/12/attack_trees.html

[3] https://www.us-cert.gov/bsi/articles/knowledge/attack-patterns/attack-pattern-usage

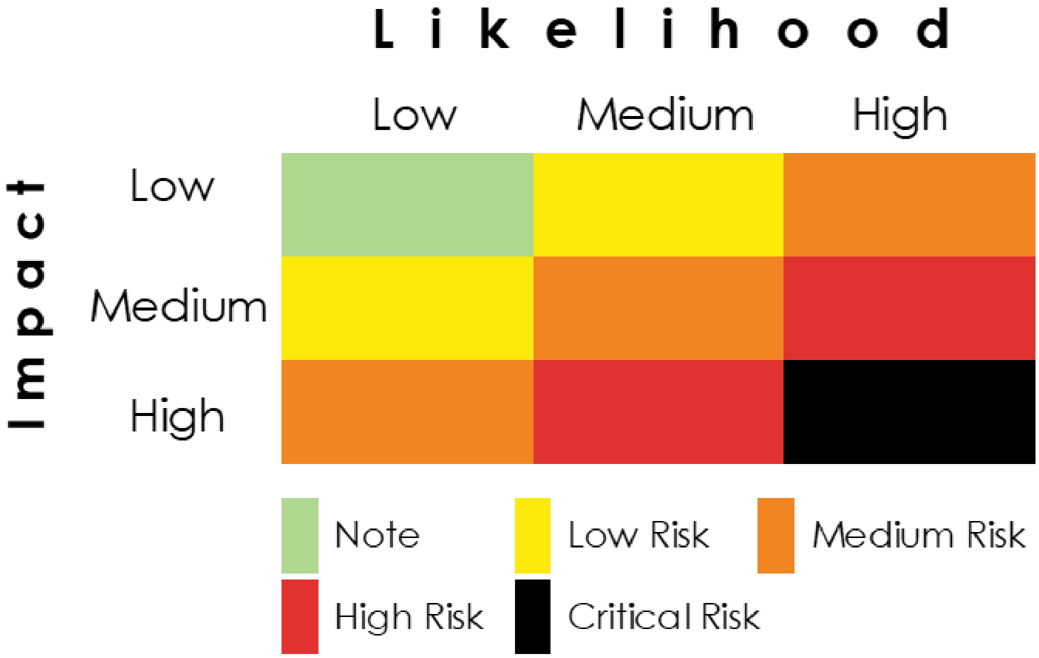
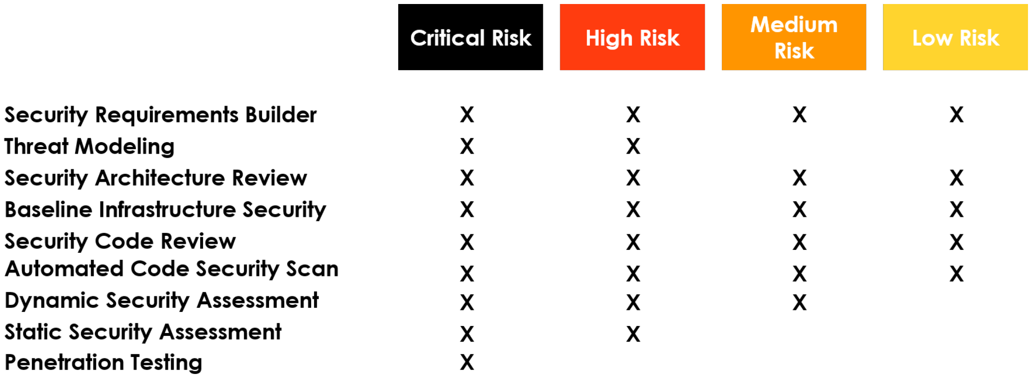


Figure 6.2: OWASP Risk Classification

"*view*" booking details, resulting in a low impact. The overall assessment is "*Medium Risk*".

This example is missing context, however, it just illustrates how the model can be used. Each organization can develop its own way of determining the impact based on standards, regulations or market competition. Once the security classification is set, the next step is to review what security actions are to be followed in the following stages as shown in Figure 6.3. Again, these actions are best determined within the team based on the application, the data it is processing and the context it is working on.

The final result of this step is a full understanding of the security implications of each item on the functional requirements planned for the sprint and what actions to take to mitigate these implications. Once that is finished, the team can proceed to the following steps which can be followed fully in case of a critical risk or partially in case of other types of risk.



	Critical Risk	High Risk	Medium Risk	Low Risk
Security Requirements Builder	X	X	X	X
Threat Modeling	X	X		
Security Architecture Review	X	X	X	X
Baseline Infrastructure Security	X	X	X	X
Security Code Review	X	X	X	X
Automated Code Security Scan	X	X	X	X
Dynamic Security Assessment	X	X	X	
Static Security Assessment	X	X		
Penetration Testing	X			

Figure 6.3: Risk Classification Action List

6.2.3 Threat Modeling

Threat Modeling is an approach of specifying security requirements, however, it approaches the system from the attackers perspective. In threat modeling, threats are identified by creating scenarios of possible inside or outside attack, with the goal of compromising systems' assets such as data or processes [88]. Threat modeling includes four main steps according to Ingalsbe et al [62]. First, defining the scope which entails defining the target processes and/or data sources. Second, the target is to be modeled using Data Flow Diagrams and the roles of the involved stakeholders are defined keeping in mind the business objective of the process and the uses cases. Then, a tool can be used to generate threats that are assessed by the team and possible mitigation approaches are defined. Finally, the threats are documented for the following stages [int.11] [int.13] [int.15].

6.2.4 Security Architecture Review

On an enterprise level, where multiple DevOps teams work concurrently on the same application, security architecture becomes as important as the software architecture. For these teams, security is defined by the main initial baseline security covered in 6.1. However, within these guidelines, each team will have their own interpretations, resulting in a misaligned security within the same application. The security controls used in each increment is mapped to embrace the business value. By security architecture review, the team ensures that similar security controls are enforced by all teams. These reviews are also used to update or change the initial baseline security when required [int.10] [int.15].

> "*Enterprise security architecture is based on what is the business objectives and you map that all the way down to security controls they based on the business impact assessment and the application ... it all provides and security architecture reference, which shows which controls to be implemented.*" [int.10]

6.3 Development Environment

Development environment is where developers translate business requirements into a working software and build the code of new features of big fixes. A developer has a main role in creating a working secure code for every requirement keeping in mind the newly generated threats. He also takes the responsibility of discussing issues with the team during feedback sessions such as the stand up meetings in scrum. Security issues become an integrated part of the requirements as any other functional requirements that can be discussed when needed. There are two levels of abstraction in the development environment, the first one is on an individual level per developer. The main concerns are how to ensure that a developer is developing a secure code within the integrated development environment (IDE). This includes a code that is generating vulnerabilities or is using outdated open-source libraries. The second level of abstraction is on a team level as group of developers how do we approach security and apply security guidelines for each sprint. This refers to group feedback and reviews on flagged issues and updating/improving the guidelines. Four main components are named in this environment: secure coding, coding guidelines, static validation and dynamic testing.

6.3.1 Secure Coding

Developing a secure code is easier said than done. A developer have to think twice about the code, first how to deliver a business value or solve a problem then think about how to do that securely. When creating an API, a developer would create stubs that temporarily substitutes the other application the API is communicating with. Stubs are created to simulate how the created API would interact and handle communication with other applications. Once ready, these stubs are to be removed. Similarly, a developer would hard-code session tokens, private keys or even passwords to test if the application would work if security factors are not in place. The real harm takes place when forgetting to remove these from the code before committing the code. An attack can gain access to the system and authentication schemes will fail because of one, yet deadly, bad practice [int.05].

Secure coding is a learning process. A developer, as a team member, is participating in planning the sprints. Therefore, he knows about the threats each sprint is introducing. Addressing these threats requires knowledge and skills to produce a working secure code. It is mentioned in section 5.2.5 that a team needs to go through a learning process and secure coding is one of the main areas where the team has to learn. For teams who are starting with DevSecOps, forming a secure code academy can help not only developers, but also the other team members on how they approach secure coding and how to see applied in each sprint. The academy is a set of trainings that encourages the team to discuss their own security-related mistakes. Another approach, which can be used in parallel with the secure coding academy, is the security champion model [15]. In this model, one developer, an ambassador, is trained on secure coding and then he would act as a reference point for all within the team with questions regarding software security. Such an approach depends on the level of cooperation of the team members.

6.3.2 Coding Guidelines

Secure coding guidelines is not a long list of bad coding practices. It is an application-specific, interactive documentation of coding best practices that emerges from the secure initial security baseline 6.1 and coding 6.3.1. It is also governed by regulations and standards. The documentation is updated at each cycle and revisited during the planning and coding stages. The team needs to be always refreshed about these guidelines.

Although each application will have its own context and requirements, there are a general guidelines that can be the starting point at the beginning of new projects. One example of these guidelines is the OWASP secure coding practices[4]. In this document, there are different checklist-type of guidelines to cover input validation, authentication and authorization, session

[4] https://www.owasp.org/images/0/08/OWASP$_SCP_Quick_Reference_Guide_v2.pdf$

management, file management, database security and many others [int.15].

6.3.3 Static Validation

The first real test to security starts in the development environment when it is easier and cheaper to spot and fix vulnerabilities. In static validation the code is not executed [36]. A tool is used where the input will be the source code and the output will be security vulnerabilities, if exist. Such tools are configured per application and in some cases, a security specialist needs to jump in and explain the bug and why it could circumvent security [95]. One example of these tools is SonarQube[5]. The new approach of these tools is to be integrated to the IDE and give developers instant feedback on possible issues. This type of tests can be fully automated. Instead of long session of traditional security code review, a tool can instantly crosscheck all the code as per the defined rules.

> "*Automation enables you to preserve consistency among all environments and you are sure that all your systems are at the same level of security and no manual human intervention involved that could create problems.*" [int.16]

Following Fairley's understanding of static validation [36], the type of issues discovered at this phase fall into two main categories. Assuming that security requirements are explicit, complete, concise, understandable, and unambiguous, the first issue would be not following the security requirements as detailed in the planning, with no justification of doing so. The second category is missing logical flow in coding the security requirements. With a properly configured security static analysis tool, both of these issues will be discovered and reported to the developer [int.07] [int.09] [int.14].

6.3.4 Dynamic Testing

Once the source code is bug free, it is time to test the compiled application. Dynamic testing examines the code after being executed against security attacks such as injection and cross site scripting. The tests observe how the application is going to react [39] and the way an application handles specific behavior from users. The dynamic tests are application specific tests that

[5] https://www.sonarqube.org/

are built along with the application to ensure that an application can handle most types of user behavior without breaking the application or causing unwanted results. These errors are not necessarily issues with the code but rather how a software user can behave. For example, tests that check if a user can access the admin panel by tampering URL values. If the test succeeded, it is interesting for the tester to see how the application responded by for example sending a notification to a monitoring application. These types of tests are application-specific and a tool can not identify what a user can or can not do [35], hence the tailored security tests. However, they execute automatically [int.10] [int.15].

In software testing, static and dynamic tests are often combined. Elberzhager et al. [33] did a systematic mapping for combining static and dynamic testing techniques. They found two main categories for combining static and dynamic testing. The first one is *compilation* and it implies that static and dynamic tests are carried out separately. The second one is *integration* and this implies that static testing results are used as input for dynamic testing. Under each of these categories, sub categories are defined that describes in more details each technique.

In this framework, dynamic testing is an automated set of tests that can be conducted autonomously or collectively. They can be implemented in the development environment or in the testing environment or in both.

6.4 Testing Environment

The testing environment receives all the developed code and runs all the tests required. It is normally a virtual environment created by Virtual Machine or Containerization. In this controlled environment, the compiled code that is developed by developers meets the tests prepared by testers. If any of the tests fail, the code is rolled back to developers with appropriate reporting on the issues to be fixed. For the purposes of testing dummy data is created, real data must not be used in testing.

Besides the dynamic testing that is covered in section 6.3.4, testing environment can include vulnerability scanning, fuzz testing and penetration testing. The first two tests are automated while the penetration testing depends on the depth of the test and the type of application. The following sections elaborate on each of these concepts.

6.4.1 Vulnerability Scanning

Vulnerability scanning checks a running software from the outside [35]. A web application scanner takes the URL of the target application as an input. Then it crawls to all the links looking for security issues. Finally, a report is created for the tester. There are three levels for the depth and speed of these scans, a quick scan looks into the first value of every parameter. The next level is a heuristic scan where a scanner will determine which parts of the application require more in depth scans. The last level is the extensive scan where a scanner will check all possible combination of values for the whole application. The time required to perform the scans increases as the depth of the scan increases [85]. A scanning performs only information collection and by itself it does not form any sort of attacks on the system. Additionally, there exists tools like OWASP ZAP[6] to perform this task, making it easy and efficient to have these insights on both the testing and the production environments. The reports can be used to improve the software (feedback for developers), or it can be used as input for further tests and attacks (feedback for testers) [int.09] [int.12].

6.4.2 Fuzz Testing

Fuzz testing, or fuzzing, is a type of dynamic test that examines a running application by generating random input values with the aim of breaking the software or disclosing a security vulnerability [48]. New fuzzing tools are based on intelligent models test, making it easier to use these tools by non security specialists [94]. Fuzzing is differentiated from other types of dynamic tests as it is powered by tools that can be customized for each application. In other words, the whole test is not developed from scratch but instead it is generated from a tool. One of these tools is the JBroFuzz[7], which is another tool developed by OWASP [int.11].

6.4.3 Penetration Testing

Penetration testing, or pen test, is the highest level of security tests. It is expensive and preparations for one test can take a long time, yet, it is the best way of finding more complex vulnerabilities. Penetration testing is more of an

[6] https://www.zaproxy.org/

[7] https://www.owasp.org/index.php/JBroFuzz

art than science and it depends on the skills of the pentester to find security issues [74]. Penetration is defined as a "*process of simulating attacks on a system in order to detect any potential vulnerabilities, and usually confirm their existence by exploiting them*" [79]. The people who perform these tests are ethical hackers and they are granted permission from the application owners to perform such attacks [int.06] [int.09] [int.11] [int.13].

Alam et al. [37] explained the four steps to be conducted in a penetration testing as shown in Figure 6.4. First, a *planning* step takes place where the goal and the scope of the test are defined. The goal for the test can be to have access to admin panel or have access to the database. Creating a concrete goal keeps the test and the testers focused on a clear goal. On the scope level, the team decides if it will be a black-box where the attacking team does not know much about the software, or a white-box. Additionally, in the scope it is determined if the operations team will be notified about the test, or not. Deciding these points manages the expectations of the involved teams. The next step is to run a *discovery* process. The attackers will run vulnerability scanning tools on the software for the purpose of exposing potential entry points. Additionally, they can also send phishing emails, perform phone calls (social engineering) and approach the staff, aiming for an entry points. All the information gathered is then to be analyzed by the attacking team to find weak points for the next step. The next step is to *attack* the system using the information gathered while keeping the goal of the test in mind. During the attack, new vulnerabilities might be revealed, these can be used also to go further with the attack. The last step is to document all findings and *report* back to the team.

Penetration testing can be risky and result in damaging the system. It is possible to perform this on a testing environment, however, performing penetration on production environment simulates what a hacker would really do. Based on the sensitivity of the application, an organization can decide on which environment this test shall take place.

6.5 Production Environment

In DevOps, production environment represents the end environment where all code is deployed. It is the environment that is accessible for users and it contains all the users' data. The production environment is a busy environment. Although the focus here is on security, in production many players are

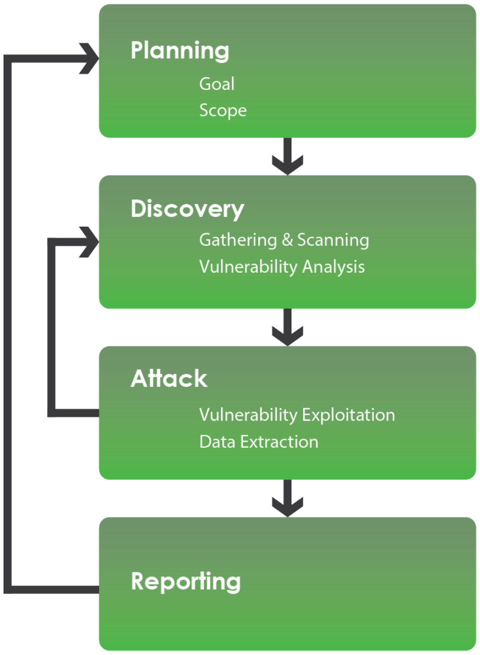


Figure 6.4: Penetration Testing Steps [37]

involved to handle all types of maintenance, load balancing and performance, new deployments, business intelligence and monitoring. Each item of these mentioned tasks has a team and a whole list of sub categories with specific goals. The idea here is that many teams are involved, making security of all these areas is also crucial for the overall security of the application. For high availability application, conducting tests on production can be a disturbing idea in case of a test failure [102].

For security, the production environment has the most risk as it has the actual user data and it is facing the outside threats. Securing this environment starts by securing the infrastructure and the network where the application is working. For the application itself and following the security checks that took place in the previous steps, many security measures can take place. Starting with security monitoring where a dedicated team keeps an eye on all security irregularities. Then, a configuration check is performed on all application servers to ensure consistency of configuration. Additionally, in the production environment, it is of high importance to create a traceability logs that can identify the changes made in the application or application configuration. Besides, there are measures that can continue in production, namely: vulnerability scanning and penetration testing.

6.5.1 Security Monitoring

Although DevOps is advocating for end-to-end ownership, monitoring is one of the areas where a centralized team is responsible for monitoring the whole application or stack of applications at once. Monitoring requires looking at the whole picture thus making it pointless to monitor individual features. Monitoring is conducted in a Security Operations Center (SOC), that looks into a stack of applications. They are responsible for spotting any malicious activities and contacting the right team in case of an incident. Although monitoring is performed in the last stage of DevSecOps, it is well thought of at the start of each new feature. The planning includes how a feature will be monitored and reported [int.05] [int.07] [int.12] [int.15].

> ".. *detect, that is monitoring where you are seeing what is happening in your application or product. Normally, you don't do that on one application, but companies do that on a SOC (Security Operations Center) for all applications and infrastructure.*" [int.09]

Monitoring, in its broader term, is about observing the application and identifying if the application is performing the way it should. However, using the right tools which are configured correctly is essential in having effective monitoring [89]. Monitoring includes logging and logging reviews. Logs are activities that happens within the application such as creating new accounts, creating admin accounts, accessing the database or failed authentication. These logs can flag suspicious activities or in worst case scenario it will show attackers trails after an attack. The level of details in these logs depends on the application and regulations. For one application, logging failed authentication can be enough while in others, the IP address of the request is documented as well. Logging information can hold very valuable information, making them a target on their own. At the same time, having a long lists of logs is not the goal, a tool or a person has to look into these lists and flag irregularities. Following a flagged log, there must be a course of action to take place and that is incident management. A security incident is defined as a "*single or series of unwanted or unexpected information security* *events that have a significant probability of compromising business operations and threatening information security*" [98]. The incident management has a whole process of classifying the severity and urgency of an incident and escalating actions until it is solved. Incidents are also a stream of feedback that can help improve the application's security in future iterations [14].

6.5.2 Configuration Check

In production, configuration is a big task. It is synonymous to develop a good security system and then leave the front door open. Although this should not be a problem, it is one of the top security breaches on the OWASP list number 6. Configuration complexity grows with the application. Configuration involves policy and management view that is eventually applied on the whole system [8]. Failing to translate configuration correctly can result in security vulnerabilities that is used to compromise the system. Configuration can be seen at an application level, where an operator manages the configuration within the application, for example expired user's ID. It can be also seen at an infrastructure level that an operator manages the flow of the software in the pipeline to perform the pipeline tasks such as build, test and deployment [4] [int.12] [int.14].

6.5.3 Traceability

Traceability is part of monitoring. It implies the ability of tracking changes within the software. In case of security incidents, the security team will be able to find the traces of the attack and identify the vulnerability used in the attack. Goa et al. [42] defined five main traceability components. 1)*Operational trace* registers the interaction between the software components, such as function calls. 2)*Performance trace* records performance data and benchmarks which is used to define data-flow congestion and to tune the software performance. 3)*State trace* reports the state of a component or data. 4)*Event trace* documents the sequence of changes or the interaction with users in the software. 5)*Error trace* records the generated errors or exceptions which are reported back to the team.

> "*...it is important to be able to trace an incident. You need a unique identifier to be able to determine what applications in-*

> *volved in this process, traceability. That is also a form of security...*" [int.14]

6.6 Infrastructure Security

In DevOps, infrastructure is an enabler for software development and operations. All the day-to-day tasks take place in one of the main environments. Each team member has a certain role within a specific environment and preparing the infrastructure required for development is part of the tasks handled by the team. Infrastructure refers to the application's requirements to run throughout its life cycle such as servers, virtual machines or containers. This includes three main tasks, first preparing and connecting required machines physically or virtually. Second, install and configure the required software for the application. Finally, install and prepare auxiliary services that is required by the application [2]. There is a big room for automation in infrastructure by developing scripts which are to be used and reused. The notation used for that is *Infrastructure as Code* (IaC). It should be noted that, although cloud is currently a trendy technology and most deployments would be on cloud, infrastructure includes private clouds or on-premises servers. The additional risk, that private clouds and on-premises servers bring, is controlled by physical access, operating system, network configuration and application vulnerabilities [60]. The debate of which solution is best can be tricky and influenced by regulations, management strategy and application risk.

In all cases, infrastructure type will always have additional risk that are type-specific. However, in general there are main threats that can be looked at regardless of the type of infrastructure used and these are environment segregation, vulnerability mapping, patching and update management, access management, system-wide traceability and encryption. The following is a detailed description of all these concepts.

6.6.1 Environment Segregation

In DevOps, many environments are created for different purposes. While here the focus was on development, testing and production environments, organizations can have their own environment setting. In some cases there will be an acceptance environment, where all the ready code is stored to be pushed to production environment. In other cases, an application is dependant on a third party application, therefore a special environment is created for the vendor to troubleshoot problems. Environment segregation is a feature for containing contamination or a security breach. Although attacks will be mainly targeting the production environment as stated earlier, a successful attack on development or testing can allow attackers to tamper the code by installing a backdoor before deployment. Creating dedicated environments for each stage with proper authentication minimizes the risk of creating attacks at the source code. Additionally, when circumventing one of the environments, the damage is limited to that environment from an infrastructure level. Finally, using multiple environments makes it easier to destroy the environment once its task is completed.

6.6.2 Vulnerability Mapping

New vulnerabilities are reported everyday. Either it is reported on a public vulnerability database, or it is discovered internally, these vulnerabilities increase the level of risk of an application [53]. On the application level, static analysis can disclose such vulnerabilities. However, on the infrastructure level, many third-party software is used. Starting with the operating system, moving down to the server services, firewalls and applications, DevOps uses a pipeline that is fully dependant on such tools and applications. For example, Apache server[8], which is a widely used web server, has a known vulnerability list. These vulnerabilities can be used to initiate attacks. Thus, knowing and mapping such vulnerabilities is helpful in understanding the risk of such tools and look for alternatives if possible. In this example, Apache is one application. The mapping needs to happen on all of the used software.

6.6.3 Patching and Update Management

In a closed connection with the previous point, patching and update management is the next step in covering vulnerabilities. Mapping the vulnerabilities allows for an overview of the used application. In most cases, if a vulnerability is discovered, the owners of the application will release a patch that mitigates the vulnerability. Managing patches and updates is an art. A pipeline is configured with many applications in place, changes can have an

[8] https://httpd.apache.org/security/vulnerabilities$_2$4.*html*

influence on the data flow between environments. Although the applications owners will try to push fixes as soon as possible, installing these updates at the end-point machines takes weeks or months. The main reason behind that is the overflow of security patches to a point that security teams are overloaded with patches. Besides, these patches can change the way an application is working or remove a feature that is needed in the pipeline configuration [16]. Therefore, creating an update and patching processes and making use of tools to automate installation will allow the teams to quickly assist and install updates and patches.

6.6.4 Access Management

In access management, two main areas are discussed: physical access to servers and digital access [61]. Physical access refers to individuals having access to the data center or server room where they can plug external devices and have direct access to the infrastructure machines. In digital access, authentication and authorization are the main concerns. The definitions of these concepts were introduced in a previous chapter, in table 4.1. Access has many shapes and forms, the simplest is user name and password credentials that a user feeds into a login page. There are more complex approaches such as *Identity and Access Management* IAM[9] and *Open Authentication* OAuth[10]. These systems perform the administration task of granting, tracking and invalidating access to certain or all parts of the infrastructure.

6.6.5 System-wide Traceability

As discussed in section 6.5.3, traceability helps understand the trigger of issues and system errors. On the environments level, this component implies the ability to trace changes on the infrastructure such as creating new environments, containers or adding a new virtual machine to the stack.

6.6.6 Encryption

In the software infrastructure level, data has three states. The data can be stored in a database or a server and that is known as data at rest[11]. The

[9] https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[10] https://oauth.net/2/

[11] https://cloud.google.com/security/encryption-at-rest/default-encryption/

second state is when data is transferred to another server or end user, which is referred to as data in transit[12] and the final state is when the data is being processed. For the first two states, applying encryption improves the security of the infrastructure. Encryption is security control that does not stop an attack, but makes the data collected useless by making it difficult or near impossible to decrypt. An attacker can target the data when it is being transferred. An example for encryption is the TLS[13] (Transport Layer Security) that aims to create a secure connection between two parties. Encryption of data at rest has also many standards that were developed over the year. The AES (Advanced Encryption Standard) is commonly used as it is easy to encrypt and difficult to decrypt. Using encryption on the infrastructure adds an extra layer of security.

[12] https://cloud.google.com/security/encryption-in-transit/

[13] https://tools.ietf.org/html/rfc5246section-2

Chapter 7

Validation and Improved Artifacts

This chapter elaborates on the improvements that were introduced to the artifacts in the previous chapters. For validation, a focus group was conducted to improve the developed artifacts. The details of the focus group setting is explained in section 3.7. The following section highlights the main important improvements that were discussed in the validation session with an illustration of the final proposed artifacts.

7.1 Impact Areas

The impact areas did not face a significant change. The discussion was about why there was no process items introduced. The main reason of not including these in the initial version was that, all the processes are represented by the framework which is the second artifact. The connection between the two artifacts was not clear, therefore in the final version the stages of DevOps have been added to the processes block in the impact areas model. The final model is presented in Figure 7.1

7.2 DevSecOps Proposed Framework

Unlike the first artifact, the framework had a couple of significant improvements. These changes are listed below. Besides these changes, there were

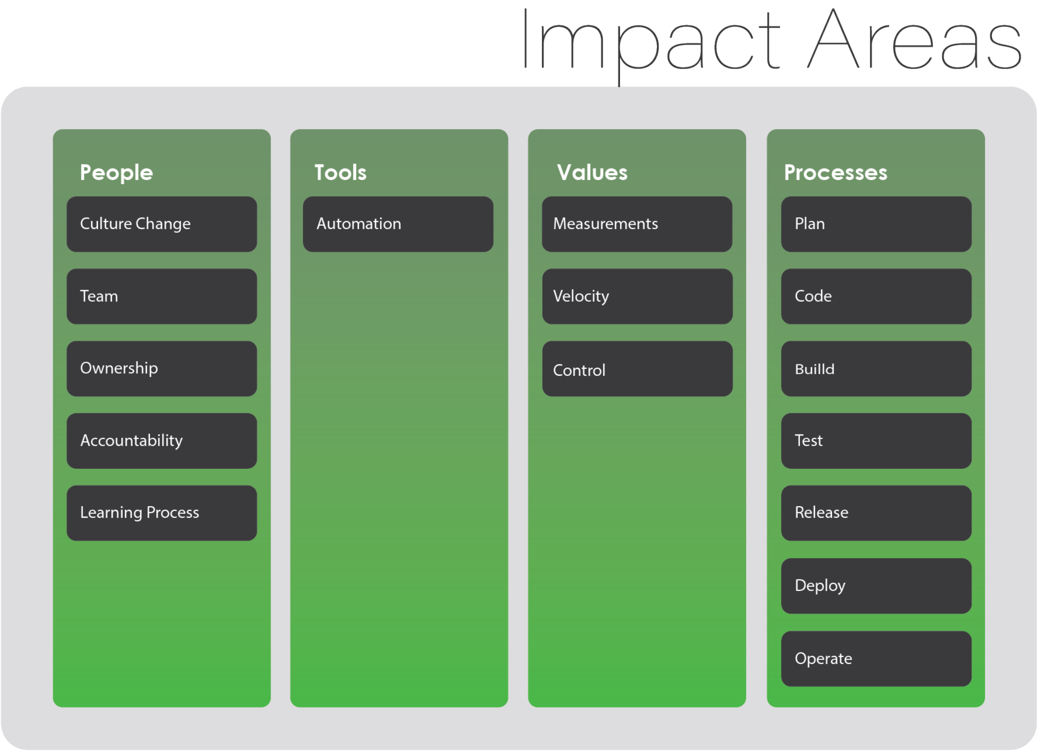


Figure 7.1: DevSecOps Impact Areas

minor changes in naming and ordering of items. The final framework is presented in Figure 7.2.

- **Adding one more step before the initial security baseline:** this point was raised when talking about the coding guidelines which are discussed in section 6.3.2. The guidelines themselves are developed in an earlier process, preferably in the initial baseline security. Further, the initial baseline security is derived from regulations, policies and standards. These regulations can be local regulations of a country or group of countries, such as the GDPR. They can also be specific to a sector, for example the financial sector, or the health sector. Moreover, policies are a reflection of the organization's strategy. They are internal documents that embrace values within the organization. Standards are developed by specialized organizations to give validity and reliability to products. Thus, based on the application context and the regulations,

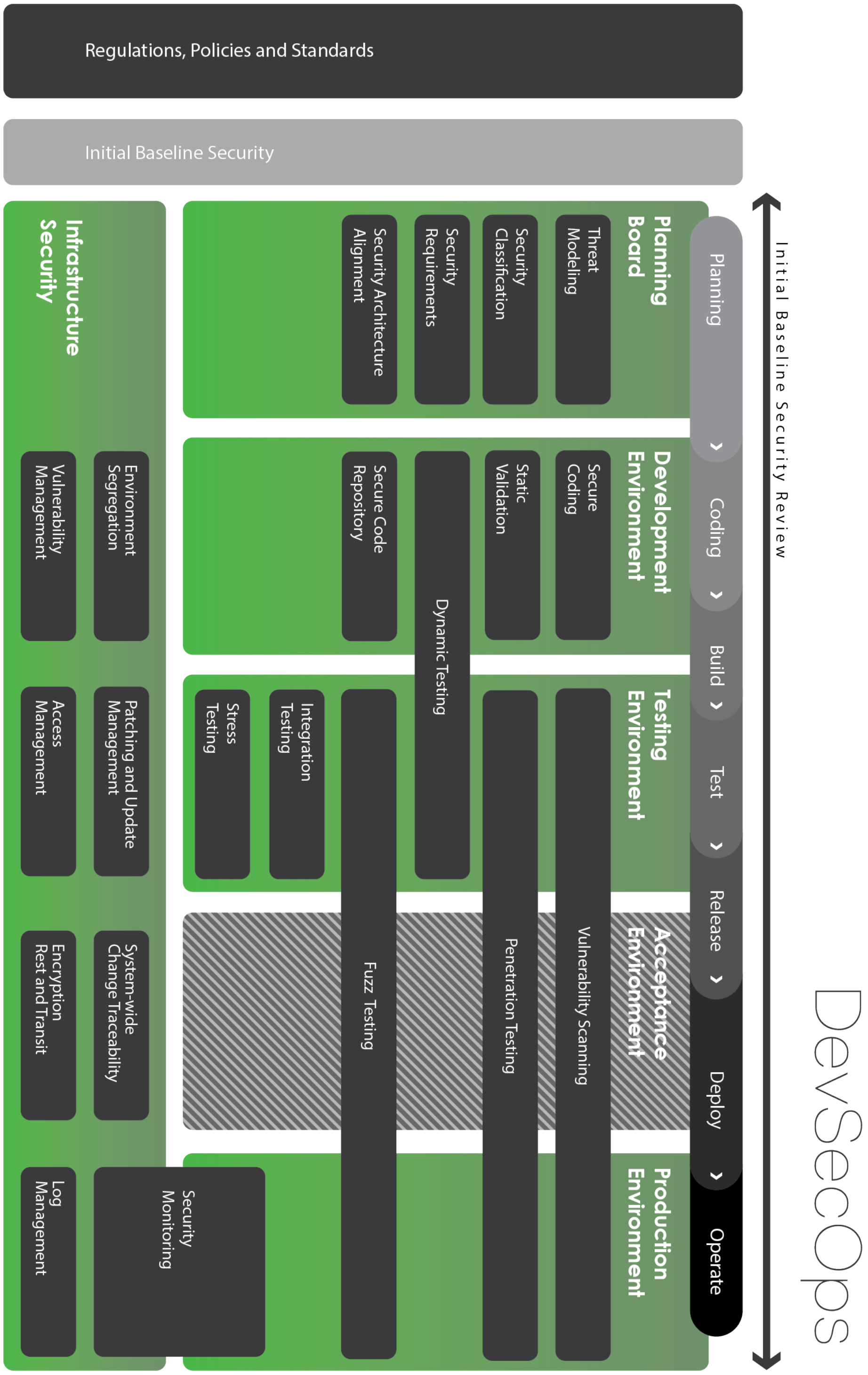


Figure 7.2: DevSecOps Proposed Framework

policies and standards influencing the context the initial baseline will be created [part.22] [part.18].

- **Adding secure coding repositories:** this concept is added to the development environment. When coming across a problem while coding, a developer will use the internet to find a solution which can be a library or a code sample. The code will be cleaned before it is used in the software. The idea of this item is to create an internal repository for the cleaned code so team members can used it again. Such repositories allow the team to save time on looking for clean secure code [part.22].

- **Integration and stress testing:** integration testing is a quality assurance method to test how the different components of an application are interacting. There are three main techniques, increment integration, top-down integration and down-top integration [97]. From a security perspective, one increment can be secured individually, however, when this increment is used with another increment, a vulnerability is created. A stress test, on the other hand, handles how the system is behaving when it is under heavy traffic. In a microservice architecture, if a service is stopped because of a DDOS attack, that could create a vulnerability in the system. Stress tests creates scenarios about what to do in case a failure happens [part.17].

- **Adding the acceptance environment:** in practice, acceptance environment is used as a staging environment. It is a type of repository where all the tested code that is ready to be pushed is placed. This environment is used for handoffs and final approvals from the higher management or stakeholders. This bridging environment has no activities that can not be done somewhere else. In an ideal DevOps pipeline, increments that are fully developed and successfully passed testing can be directly and automatically pushed to the production environment, that is known as end-to-end DevOps. This approach requires a high level of maturity and trust within the team. The acceptance environment breaks this idea by requiring an approval before pushing new increments. Although it is not a recommended approach, it is applied widely in practice, therefore it is added to the model in shaded colors as it can be skipped [part.19].

- **Monitoring of application and infrastructure:** while, it was discussed in section 6.5.1 that monitoring includes many concepts, monitoring includes even more concepts. The traceability and configuration check were considered as parts of monitoring. Further, the power of monitoring comes from the overall view with meaningful monitoring metrics. It is more effective and efficient to perform monitoring over all the infrastructure and the stack of applications all together. What should be monitored is specific to the context of the application, keeping in mind how useful the information on the dashboards and if we are looking at the correct parameters [part.21].

- **Adding log management:** finally, logs that are created from events on the infrastructure need to be looked at and processed. There will be no meaning in creating a log with all types of information, if there will be no one to look at them and elicit patterns, significant events or malicious activity [part.21].

Chapter 8

Discussion and Limitations

This chapter discusses the main concepts in the thesis and articulates on the artifacts presented in previous chapters. The second part talks about the main limitations of the research and how they were mitigated.

8.1 Discussion

DevOps is being implemented widely in the past few years. Organizations can see and feel the value DevOps is adding on their software product. However, security, which is a different team, has a role in creating a secure application. Yet, their involvement in the process is very limited, if not totally absent. In the best case scenario, the security team will be involved from the start by defining required security controls and then test them on a later stage before going to production. Breaking this silo requires shifting security responsibilities to the whole team and trying to produce a secure product from the beginning instead of adding it on top.

Including security within a DevOps team is easier said than done. DevSecOps does not mean to have three teams talking three different languages, rather than one team that takes security into account in each step. It requires a cultural change and a mindset that takes security seriously and acts on that. The culture of DevSecOps extends the DevOps culture by introducing security as an integrated part of the software and acknowledging that security requirements are not less important than functional requirements. The team dynamics and knowledge must allow the team to work seamlessly in producing secure code. When security issues are raised that are beyond the capabilities of the team, experts are consulted, however, these should be the extreme cases. The normal case is that a team has sufficient knowledge and training to handle most of the security issues. In practice, experts look into security as an essential part that must be planned more thoroughly in the beginning. Checking security requirements only at the testing stage is risking hours of work and failing on the testing stage costs more to fix compared to addressing these issues on an earlier stage.

The artifacts of this research are first steps in defining the outline of DevSecOps. These artifacts help organizations understand what this topic is about and the core concepts to think about when trying to implement these concepts in practice. Scoring all of the components in the framework for every increment is an overwhelming task. Thus, planning and classifying increments is the first step to achieve reliable security with reasonable resources.

8.2 Limitations

This section covers the context and the boundaries of the research. Three main areas are relevant to this research as defined by Yin [106], contract validity, external validity and reliability. Additionally, the last section talks about the evaluation limitation.

8.2.1 Construct Validity

Construct validity tests if correct operational measures were followed to arrive the claimed conclusions. In this research, several actions were taken to ensure construct validity. First, this research uses multiple sources such as interviews, literature and other documents. For interviews, interviewees were grouped in three main groups, development, security and operations to cover all points of view. In analysis, triangulation is used to cover all possible components. Moreover, the interviewees came from different levels and roles, where high level interviewees focused on strategic and enterprise components and low level interviewees focused on the team component. The constructs where validated in a focus group where the main questions where how correct and complete the models were.

8.2.2 External Validity

External validity asks the question if the results can be generalized. In the context of this research, the short answer is no for two main reasons. The research was conducted in an organization, Accenture, where all the interviewees are working. Although the interviewees work with a wide variety of clients and this research is supervised by academic supervisors, the main content of the research can be influenced by the culture of the organization. Generalizing the results would require conducting the case study in different organizations. The other reason is related to DevOps itself. At the start of the research, it was discussed that DevOps has multiple definitions and organizations can develop their own understanding of DevOps. In this research, the focus was on one of these definitions and one possible pipeline configuration. While many concepts are still the same across different types, adjustments to the framework will be required based on the DevOps implementation.

8.2.3 Reliability

Reliability refer to the possibility of repeating the same research setting and reaching to the same conclusions. In order to ensure reliability, the protocols are documented and enclosed in the appendix. Conducting the same steps will result in the same results given the research environment.

8.2.4 Evaluation

The method used in this research was Design Science Research. As indicated in section 3.2.8, this method is used to solve a real-life problem. For this purpose, the artifacts were created and validated with experts. However, due to the limited time of this research, the results were not applied in a real-life case. This step can be a start for a new research.

Chapter 9

Conclusion

This chapter covers the answers to the sub-research questions and the main research question. The main research question was "*How can security be added to a DevOps framework to guarantee continuous security and rapid value delivery?*". The main question was trying to shape the understanding of the DevSecOps concept and how that can be joined in the current DevOps pipeline. The sub-research questions focused on understanding what DevOps means and what security controls can be highlighted in DevSecOps pipeline along with main changes to team composition and tools. The following is a detailed answer to all of the questions posed in section 2.

9.1 Sub-Research Question 1 Answer:

How is security approached in DevOps framework in theory and in practice?

This sub-research question is answered by conducting a literature review and interviews with practitioners. DevOps by its definition can have many variations based on the organization, application and local regulations. DevOps as a concept has a sufficient literature that not only describes abstract concepts like culture, value and team qualities, but also details the processes and tooling. Security, on the other hand, has a well-established literature that details security controls at the application layer. In practice, many organizations have already reached an advanced level of DevOps maturity in the Dutch market. The same applies to security. Yet, the integration of security within DevOps is missing. Currently, most organizations are not full end-to-end automated DevOps. In most cases, there will be a management approval for new deployments. Security is handled as a different department that is consulted when security approvals are needed to deploy. DevSecOps as a concept is still new and there is limited scientific research around it, yet many practitioners acknowledge the value of including security within DevOps. The findings of this question are reported in section 4.

9.2 Sub-Research Question 2 Answer:

What is the impact of implementing DevSecOps on organizations?

This sub-research question is answered by conducting a case study. The initial result of this question is visualized in Figure 5.1. The impact of integrating security within DevOps has four main areas, people, tools, values and processes. These main areas include other core concepts such as culture, team, ownership, automation, control along with the DevOps stages. The first impact area is people, which refers to the persons involved, mainly the team. It was discussed how would they approach this new mindset and how does it change the current way of working. The change also impacts the setting of pipeline and the tools used to create the pipeline. New tools have to be introduced to perform the security related tasks. The third area is the values and that represents the business side. This area discusses the added value DevSecOps brings to the team and the organization. Finally, the processes, which is the DevOps stages, have changed. The change is detailed in the following sub-research question.

9.3 Sub-Research Question 3 Answer:

What are the changes that DevSecOps introduces to a DevOps team and how will those changes impact a DevOps team?

This sub-research question was also answered by the case study. The initial results are presented in section 6. This framework shows the extra components that need to be considered to include security in DevOps. The framework does not propose changes to the DevOps stages, but rather adds extra components for each stage that brings security into focus. The framework has three main areas, the DevOps stages which are represented by the corresponding environments, the initial baseline security and the infrastructure security. There are security controls that can be applied at each of these areas.

9.4 Sub-Research Question 4 Answer:

How are the proposed artifacts evaluated by experts?

This last sub-research question aimed to improve the initial artifacts. It was answered by conducting a focus group session where the initial models were presented and improved. The results are presented in section 7. For the impact areas, no major changes are introduced. However, for the framework a number of improvements were discussed. First, an acceptance environment is added, although it is not ideal, but it is applied widely in practice. Monitoring is performed on both the infrastructure and application levels at the same time. Further, smaller improvements in naming were discussed. The validation itself was positive and the experts thought it is a good start for understanding the DevSecOps topic.

9.5 Main Research Question

How can security be added to a DevOps framework to guarantee continuous security and rapid value delivery?

The main research question can be answered as follows. Security is seen as an enabler not a dependency. Adding security to DevOps empowers the team and helps producing secure application that have security built-in not bolted on. Including security within the team increases the agility of the team. The team will produce better software and they can do that faster. Security controls are divided along the pipeline aiming to catch security issues as early as possible where they are easier to fix. The artifacts produced in the research give a high level overview of the main areas to look at when deciding to implement DevSecOps. However, they can be further tailored per application or per context. A customized version will include the specific detailed processes which in turn will be used to configure the pipeline.

Although an automated end-to-end pipeline is always recommended, organizations will start with small changes to the pipeline and improve their automation coverage overtime.

Chapter 10

Research Contribution and Future Work

This section describes how this research is contributing to the body of knowledge and explores the areas of future research around the topic of DevSecOps.

10.1 Research Contribution

This research aimed to understand the context of how security is approached in highly rapid teams that use DevOps. DevOps has proven how value can be delivered to clients in short time spans. The goal was to understand what DevOps is, what security is and how these two can play in one team called DevSecOps. The knowledge in this research is collected from both literature and practice. The first contribution of this research is the *Impact Areas* model 7.1. This model illustrates the main areas that will be touched when trying to implement DevSecOps. Understanding these areas helps in identifying the required changes in culture, team, tools and processes. The second contribution is the DevSecOps framework 7.2. *The framework* illustrates the main components of including security in DevOps and how to distribute the effort of security across environments aiming to catch vulnerabilities as far to the left as possible where it is cheaper to fix and patch. Nevertheless, the framework is a high level abstraction that can be tailored per organization and per application. In fact, understanding the context, where DevSecOps is to be implemented, is very crucial in specifying the components of each item in the framework.

10.2 Future Work

The first step for future research is to apply the framework in a context and observe the results. According to the method used for this research, design science research, the artifacts should be applied in the environment. This can be in a form of a case study, where a team is trained for the new concepts and then apply the artifacts in three to five sprints. During this time important KPIs, such as velocity and broken builds, are closely monitored. According to the DevSecOps definition, the first few sprints might reduce the velocity, however the learning process will build up and velocity goes back to normal. The results of this experiment and study are used to improve the artifacts further.

Another vital research area is to include the other stakeholders. In the context of this research, the main focus was on the development, security and operations. However, software production has more people involved in the process. First, from an organizational perspective, the higher management and organization strategy can influence the direction of how DevSecOps is to be shaped. On the other side, the customers' needs and market requirements can have their influence on security. For some applications, such as financial and health applications, adhering to local regulations must be taken into account when designing the security of the application. In this research, DevSecOps was taken in isolation for simplicity and in order to introduce the first ideas about DevSecOps, however, taking into account other stakeholders is needed to get the full picture.

The artifacts presented in this research include general concepts. For organizations that require security support, translating these concepts into detailed processes that are fitted per application. The processes will include tooling and pipeline configuration as well as areas where automation is possible. Besides, DevSecOps is about the journey towards secure application. A maturity model gives organizations a view of the current state in security and where to head next.

Bibliography

[1] Andrew van der Stock, Brian Glas, N. S. T. G. The ten most critical web application security risks. *OWASP Foundation* (2017).

[2] Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., and Tamburri, D. A. Devops: introducing infrastructure-as-code. In *Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on* (2017), IEEE, pp. 497–498.

[3] Artac, M., Borovssak, T., Nitto, E. D., Guerriero, M., and Tamburri, D. A. Devops: Introducing infrastructure-as-code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)* (May 2017), pp. 497–498.

[4] Bartusevics, A., Novickis, L., and Leye, S. Models and methods of software configuration management. *Applied Computer Systems 17*, 1 (2015).

[5] Bassil, Y. A simulation model for the waterfall software development life cycle. *CoRR abs/1205.6904* (2012).

[6] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. Manifesto for agile software development.

[7] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. Manifesto for agile software development.

[8] Bellovin, S. M., and Bush, R. Configuration management and security. *IEEE Journal on Selected Areas in Communications 27*, 3 (April 2009), 268–274.

[9] Bouquet, F., Peureux, F., and Ambert, F. *Model-Based Testing for Functional and Security Test Generation.* Springer International Publishing, Cham, 2014, pp. 1–33.

[10] Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software 80*, 4 (2007), 571 – 583. Software Performance.

[11] Bucena, I., and Kirikova, M. Simplifying the devops adoption process. vol. 1898. cited By 0.

[12] Budgen, D., and Brereton, P. Performing systematic literature reviews in software engineering. In *Proceedings of the 28th International Conference on Software Engineering* (New York, NY, USA, 2006), ICSE '06, ACM, pp. 1051–1052.

[13] Cadariu, M., Bouwers, E., Visser, J., and van Deursen, A. Tracking known security vulnerabilities in proprietary software systems. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (March 2015), pp. 516–519.

[14] Cao, J., and Zhang, S. Itil incident management process reengineering in industry 4.0 environments. In *Proceedings of the 2nd International Conference on Advances in Mechanical Engineering and Industrial Informatics (AMEII 2016)* (2016), vol. 73, pp. 1011–6.

[15] Carter, K. Francois raynaud on devsecops. *IEEE Software 34*, 5 (2017), 93–96.

[16] Cavusoglu, H., Cavusoglu, H., and Zhang, J. Security patch management: Share the burden or share the damage? *Management Science 54*, 4 (2008), 657–670.

[17] Ceccato, M., Nguyen, C. D., Appelt, D., and Briand, L. C. Sofia: An automated security oracle for black-box testing of sql-injection vulnerabilities. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (New York, NY, USA, 2016), ASE 2016, ACM, pp. 167–177.

[18] Cerda^1, B., Martinez-Belmares, E., and Yuan, S. Protection from black hole attacks in communication networks.

[19] Chiang, T.-C., and Huang, Y.-M. Group keys and the multicast security in ad hoc networks. In *2003 International Conference on Parallel Processing Workshops, 2003. Proceedings.* (Oct 2003), pp. 385–390.

[20] Christofferson, D. *Managing Cybersecurity Risk for the Coming Decade.* Springer International Publishing, Cham, 2018, pp. 23–46.

[21] Colomo-Palacios, R., Fernandes, E., Soto-Acosta, P., and Larrucea, X. A case analysis of enabling continuous software deployment through knowledge management. *International Journal of Information Management 40* (2018), 186 – 189.

[22] Comino, S., and Manenti, F. M. Dual licensing in open source software markets. *Information Economics and Policy 23*, 3 (2011), 234 – 242.

[23] Cruzes, D. S., Felderer, M., Oyetoyan, T. D., Gander, M., and Pekaric, I. How is security testing done in agile teams? a cross-case analysis of four software teams. In *Agile Processes in Software Engineering and Extreme Programming* (Cham, 2017), H. Baumeister, H. Lichter, and M. Riebisch, Eds., Springer International Publishing, pp. 201–216.

[24] Dearle, A. Software deployment, past, present and future. In *Future of Software Engineering, 2007. FOSE '07* (May 2007), pp. 269–284.

[25] Donaldson, S. E., Siegel, S. G., Williams, C. K., and Aslam, A. *Enterprise Cybersecurity and the Cloud.* Apress, Berkeley, CA, 2015, pp. 105–117.

[26] Dong, X., Li, R., He, H., Zhou, W., Xue, Z., and Wu, H. Secure sensitive data sharing on a big data platform. *Tsinghua Science and Technology 20*, 1 (Feb 2015), 72–80.

[27] Doody, O., and Noonan, M. Preparing and conducting interviews to collect data.

[28] Dyck, A., Penners, R., and Lichter, H. Towards definitions for release engineering and devops. In *2015 IEEE/ACM 3rd International Workshop on Release Engineering* (May 2015), pp. 3–3.

[29] Ebert, C., Abrahamsson, P., and Oza, N. Lean software development. *IEEE Software*, 5 (2012), 22–25.

[30] El-Hadary, H., and El-Kassas, S. Capturing security requirements for software systems. *Journal of Advanced Research 5*, 4 (2014), 463 – 472. Cyber Security.

[31] Elberzhager, F., Arif, T., Naab, M., Süss, I., and Koban, S. From agile development to devops: Going towards faster releases at high quality – experiences from an industrial context. In *Software Quality. Complexity and Challenges of Software Engineering in Emerging Technologies* (Cham, 2017), D. Winkler, S. Biffl, and J. Bergsmann, Eds., Springer International Publishing, pp. 33–44.

[32] Elberzhager, F., Arif, T., Naab, M., Süss, I., and Koban, S. From agile development to devops: Going towards faster releases at high quality – experiences from an industrial context. In *Software Quality. Complexity and Challenges of Software Engineering in Emerging Technologies* (Cham, 2017), D. Winkler, S. Biffl, and J. Bergsmann, Eds., Springer International Publishing, pp. 33–44.

[33] Elberzhager, F., Münch, J., and Nha, V. T. N. A systematic mapping study on the combination of static and dynamic quality assurance techniques. *Information and Software Technology 54*, 1 (2012), 1 – 15.

[34] Eshete, B., Villafiorita, A., and Weldemariam, K. Early detection of security misconfiguration vulnerabilities in web applications. In *2011 Sixth International Conference on Availability, Reliability and Security* (Aug 2011), pp. 169–174.

[35] Esposito, D., Rennhard, M., Ruf, L., and Wagner, A. Exploiting the potential of web application vulnerability scanning. In *ICIMP 2018, Spain, July 22-26, 2018* (2018), IARIA, pp. 22–29.

[36] Fairley, R. E. Tutorial: Static analysis and dynamic testing of computer software. *Computer 11*, 4 (April 1978), 14–23.

[37] Farah, T., Alam, D., Kabir, M. A., and Bhuiyan, T. Sqli penetration testing of financial web applications: Investigation of bangladesh region. In *2015 World Congress on Internet Security (WorldCIS)* (Oct 2015), pp. 146–151.

[38] Felderer, M., and Fourneret, E. A systematic classification of security regression testing approaches. *International Journal on Software Tools for Technology Transfer 17*, 3 (Jun 2015), 305–319.

[39] Felderer, M., Zech, P., Breu, R., Büchler, M., and Pretschner, A. Model-based security testing: a taxonomy and systematic classification. *Software Testing, Verification and Reliability 26*, 2, 119–148.

[40] Fitzgerald, B., and Stol, K.-J. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software 123* (2017), 176 – 189.

[41] Gallaugher, J. M., and Wang, Y.-M. Understanding network effects in software markets: Evidence from web server pricing. *MIS Quarterly 26*, 4 (2002), 303–327.

[42] Gao, J., Zhu, E. Y., Shim, S., and Chang, L. Monitoring software components and component-based software. In *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000* (Oct 2000), pp. 403–412.

[43] Garn, B., Kapsalis, I., Simos, D. E., and Winkler, S. On the applicability of combinatorial testing to web application security testing: A case study. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing* (New York, NY, USA, 2014), JAMAICA 2014, ACM, pp. 16–21.

[44] Gartner. 10 things to get right for successful devsecops, 2017.

[45] Gartner. Hype cycle for application security, 2018.

[46] Gartner. Top 10 strategic technology trends for 2018: Continuous adaptive risk and trust, 2018.

[47] Ge, X., Paige, R. F., Polack, F. A., Chivers, H., and Brooke, P. J. Agile development of secure web applications. In *Proceedings of the 6th International Conference on Web Engineering* (New York, NY, USA, 2006), ICWE '06, ACM, pp. 305–312.

[48] Godefroid, P., Levin, M. Y., Molnar, D. A., et al. Automated whitebox fuzz testing. In *NDSS* (2008), vol. 8, pp. 151–166.

[49] Godse, M., and Mulik, S. An approach for selecting software-as-a-service (saas) product. In *2009 IEEE International Conference on Cloud Computing* (Sept 2009), pp. 155–158.

[50] Greising, L., Bartel, A., and Hagel, G. Introducing a deployment pipeline for continuous delivery in a software architecture course. In *Proceedings of the 3rd European Conference of Software Engineering Education* (New York, NY, USA, 2018), ECSEE'18, ACM, pp. 102–107.

[51] Gruhn, V., and Schäfer, C. Bizdevops: Because devops is not the end of the story. In *Intelligent Software Methodologies, Tools and Techniques* (Cham, 2015), H. Fujita and G. Guizzi, Eds., Springer International Publishing, pp. 388–398.

[52] Gupta, V., Kapur, P., and Kumar, D. Modeling and measuring attributes influencing devops implementation in an enterprise using structural equation modeling. *Information and Software Technology 92* (2017), 75 – 91.

[53] Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., Hamilton, L. H., Centeno, G. I., Key, J. R., Ellingwood, P. M., McConley, M. W., Opper, J. M., Chin, S. P., and Lazovich, T. Automated software vulnerability detection with machine learning. *CoRR abs/1803.04497* (2018).

[54] Hassan, M. M., Nipa, S. S., Akter, M., Haque, R., Deepa, F. N., Rahman, M., Siddiqui, M. A., Sharif, M. H., et al. Broken authentication and session management vulnerability: A case study of web application. *International Journal of Simulation–Systems, Science & Technology 19*, 2 (2018).

[55] Hevner, A., and Chatterjee, S. *Design Science Research in Information Systems.* Springer US, Boston, MA, 2010, pp. 9–22.

[56] Hevner, A. R. A three cycle view of design science research. *Scandinavian journal of information systems 19*, 2 (2007), 4.

[57] Hevner, A. R., and March, S. T. The information systems research cycle. *Computer 36*, 11 (Nov 2003), 111–113.

[58] Hofstede, G. Dimensionalizing cultures: The hofstede model in context. *Online readings in psychology and culture 2*, 1 (2011), 8.

[59] Hüttermann, M. *Beginning DevOps for Developers.* Apress, Berkeley, CA, 2012, pp. 3–13.

[60] Ibrahim, A. S., Hamlyn-Harris, J., and Grundy, J. Emerging security challenges of cloud virtual infrastructure. *arXiv preprint arXiv:1612.09059* (2016).

[61] Indu, I., Anand, P. R., and Bhaskar, V. Identity and access management in cloud environment: Mechanisms and challenges. *Engineering Science and Technology, an International Journal 21*, 4 (2018), 574 – 588.

[62] Ingalsbe, J. A., Kunimatsu, L., Baeten, T., and Mead, N. R. Threat modeling: diving into the deep end. *IEEE software*, 1 (2008), 28–34.

[63] ISO. Information processing systems – open systems interconnection – basic reference model – part 4: Management framework, 1989.

[64] Jabbari, R., bin Ali, N., Petersen, K., and Tanveer, B. What is devops?: A systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016* (New York, NY, USA, 2016), XP '16 Workshops, ACM, pp. 12:1–12:11.

[65] Kim, S., Park, S., Yun, J., and Lee, Y. Automated continuous integration of component-based software: An industrial experience. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering* (Sept 2008), pp. 423–426.

[66] Kitchenham, B. Procedures for performing systematic reviews. *Keele, UK, Keele University 33*, 2004 (2004), 1–26.

[67] Kitchenham, B. A., Dyba, T., and Jorgensen, M. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering* (Washington, DC, USA, 2004), ICSE '04, IEEE Computer Society, pp. 273–281.

[68] Kwon, O.-H., Lee, S. M., Lee, H., Kim, J., Kim, S. C., Nam, G. W., and Park, J. G. Hacksim: An automation of penetration testing for remote buffer overflow vulnerabilities. In *Information Networking. Convergence in Broadband and Mobile Networking* (Berlin, Heidelberg, 2005), C. Kim, Ed., Springer Berlin Heidelberg, pp. 652–661.

[69] Laukkarinen, T., Kuusinen, K., and Mikkonen, T. Regulated software meets devops. *Information and Software Technology 97* (2018), 176 – 178.

[70] Loise, T., Devroey, X., Perrouin, G., Papadakis, M., and Heymans, P. Towards security-aware mutation testing. In *ICST Workshops* (2017), pp. 97–102.

[71] Lwakatare, L. E., Kuvaja, P., and Oivo, M. Dimensions of devops. In *Agile Processes in Software Engineering and Extreme Programming* (Cham, 2015), C. Lassenius, T. Dingsøyr, and M. Paasivaara, Eds., Springer International Publishing, pp. 212–217.

[72] Mackey, T. Building open source security into agile application builds. *Network Security 2018*, 4 (2018), 5 – 8.

[73] Mansfield-Devine, S. Devops: finding room for security. *Network Security 2018*, 7 (2018), 15 – 20.

[74] McDermott, J. P. Attack net penetration testing. In *Proceedings of the 2000 Workshop on New Security Paradigms* (New York, NY, USA, 2000), NSPW '00, ACM, pp. 15–21.

[75] Mohan, V., and Othmane, L. B. Secdevops: Is it a marketing buzzword? - mapping research on security in devops. In *2016 11th International Conference on Availability, Reliability and Security (ARES)* (Aug 2016), pp. 542–547.

[76] Myrbakken, H., and Colomo-Palacios, R. Devsecops: A multivocal literature review. In *Software Process Improvement and Capability Determination* (Cham, 2017), A. Mas, A. Mesquida, R. V. O'Connor, T. Rout, and A. Dorling, Eds., Springer International Publishing, pp. 17–29.

[77] Myrbakken, H., and Colomo-Palacios, R. Devsecops: A multivocal literature review. In *Software Process Improvement and Capability Determination* (Cham, 2017), A. Mas, A. Mesquida, R. V. O'Connor, T. Rout, and A. Dorling, Eds., Springer International Publishing, pp. 17–29.

[78] Neely, S., and Stolt, S. Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *2013 Agile Conference* (Aug 2013), pp. 121–128.

[79] Ottosson, H., and Lindquist, P. Penetration testing for the inexperienced ethical hacker : A baseline methodology for detecting and mitigating web application vulnerabilities. Master's thesis, Linköping University, Database and information techniques, 2018.

[80] Pawar, R. G. Sql injection attacks. *KHOJ: Journal of Indian Management Research and Practices* (2015), 125–129.

[81] Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. A design science research methodology for information systems research. *Journal of Management Information Systems 24*, 3 (2007), 45–77.

[82] Ramachandran, M. Software security requirements management as an emerging cloud computing service. *International Journal of Information Management 36*, 4 (2016), 580 – 590.

[83] Rathod, N., and Surve, A. Test orchestration a framework for continuous integration and continuous deployment. In *2015 International Conference on Pervasive Computing (ICPC)* (Jan 2015), pp. 1–5.

[84] Riti, P. *Cloud and DevOps*. Apress, Berkeley, CA, 2018, pp. 209–220.

[85] Sagala, A., and Manurung, E. Testing and comparing result scanning using web vulnerability scanner. *Advanced Science Letters 21*, 11 (2015), 3458–3462.

[86] Salas, M., and Martins, E. Security testing methodology for vulnerabilities detection of xss in web services and ws-security. *Electronic Notes in Theoretical Computer Science 302* (2014), 133 – 154. Proceedings of the XXXIX Latin American Computing Conference (CLEI 2013).

[87] Sani, A. S., Yuan, D., Jin, J., Gao, L., Yu, S., and Dong, Z. Y. Cyber security framework for internet of things-based energy internet. *Future Generation Computer Systems* (2018).

[88] Scandariato, R., Wuyts, K., and Joosen, W. A descriptive study of microsoft's threat modeling technique. *Requirements Engineering 20*, 2 (Jun 2015), 163–180.

[89] Schlossnagle, T. Monitoring in a devops world. *Queue 15*, 6 (Dec. 2017), 10:35–10:45.

[90] Schultze, U., and Avital, M. Designing interviews to generate rich data for information systems research. *Information and Organization 21*, 1 (2011), 1 – 16.

[91] Senapathi, M., Buchan, J., and Osman, H. Devops capabilities, practices, and challenges: Insights from a case study. In *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018* (New York, NY, USA, 2018), EASE'18, ACM, pp. 57–67.

[92] Subramanian, N., and Jeyaraj, A. Recent security challenges in cloud computing. *Computers Electrical Engineering 71* (2018), 28 – 42.

[93] Sulatycki, R., and Fernandez, E. B. Two threat patterns that exploit "security misconfiguration" and "sensitive data exposure" vulnerabilities. In *Proceedings of the 20th European Conference on Pattern Languages of Programs* (New York, NY, USA, 2015), EuroPLoP '15, ACM, pp. 46:1–46:11.

[94] Takanen, A. *Proactive Security Testing and Fuzzing.* Vieweg+Teubner, Wiesbaden, 2010, pp. 312–319.

[95] Thomas, T. W. *Security Code Review with Static Analysis Techniques for the Detection and Remediation of Security Vulnerabilities.* PhD thesis, The University of North Carolina at Charlotte, 2018.

[96] Topper, J. Compliance is not security. *Computer Fraud Security 2018*, 3 (2018), 5 – 8.

[97] Tsai, W. T., Bai, X., Paul, R., Shao, W., and Agarwal, V. End-to-end integration testing design. In *25th Annual International Computer Software and Applications Conference. COMPSAC 2001* (Oct 2001), pp. 166–171.

[98] Tøndel, I. A., Line, M. B., and Jaatun, M. G. Information security incident management: Current practice as reported in the literature. *Computers Security 45* (2014), 42 – 57.

[99] Uwagbole, S. O., Buchanan, W. J., and Fan, L. Applied machine learning predictive analytics to sql injection attack detection and prevention. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (May 2017), pp. 1087–1090.

[100] Verona, J. *Practical DevOps.* Packt Publishing Ltd, 2016.

[101] Wettinger, J., Andrikopoulos, V., and Leymann, F. Enabling devops collaboration and continuous delivery using diverse application environments. In *On the Move to Meaningful Internet Systems: OTM 2015 Conferences* (Cham, 2015), C. Debruyne, H. Panetto, R. Meersman, T. Dillon, G. Weichhart, Y. An, and C. A. Ardagna, Eds., Springer International Publishing, pp. 348–358.

[102] Williams, C. L., McClintock, D. S., and Balis, U. G. The case for an entropic simian in your laboratory: The case for laboratory information system failure scenario testing in the live production environment. *Journal of pathology informatics 9* (2018).

[103] Wohlin, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (New York, NY, USA, 2014), EASE '14, ACM, pp. 38:1–38:10.

[104] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[105] Wotawa, F. On the automation of security testing. In *2016 International Conference on Software Security and Assurance (ICSSA)* (Aug 2016), pp. 11–16.

[106] Yin, R. K. *Case study research and applications: Design and methods.* Sage publications, 2017.

Appendices

Appendix A

Literature Review Protocol

Document Created on Week30 / 2018

A.1 Goal

This literature review aims to achieve:

- Explorer the state of the art on DevOps concepts and Security Testing.
- Position the direction of the research.
- Create the base conceptual model for this research's output.

A.2 Strategy

Collecting the papers will be through online scientific research engines. The first 10 papers for each keyword, that are published after 2014, will be collected for the literature review. The data collection is performed on week 31/2018. All the papers will go through the inclusion and exclusion criteria based on the abstract of the paper. The final list will be studied for analysis.

A.3 Search Engines

The search engines used for this analysis are:

- ACM digital library https://dl.acm.org/

- Scopus https://www.scopus.com
- Springer Link https://link.springer.com
- Science Direct https://www.sciencedirect.com/
- Gartner

A.4 Keywords

The following keywords will be used ordered from specific to general:

- DevSecOps
- DevOps
- Continuous Integration AND Delivery
- Security Testing AND Automation

A.5 Inclusion Criteria

All material will be included unless they are satisfying one or more of the exclusion criteria.

A.6 Exclusion Criteria

- Not English
- Redundancy, one version will be eventually considered
- Published before 2014
- Books
- Irrelevant to Computer Science
- Irrelevant to Software Production

Appendix B

Interview Protocol

The main objective of this interview is to identify the current state of Security and DevOps in the industry. The focus is on how security is defined and how it is approached in rapid DevOps teams. This interview is designed for DevOps practitioners and Security experts. The planned time for this interview is between 45 - 60 minutes.
For the purpose of further analysis, I want to record this interview. May I record this interview for further analysis? All information will be anonymized.
:: Start Recording ::
Just for the record, do you approve the recording of this interview?
Section I: Introduction of DevOps as a framework of developing applications.

1. Would you please briefly introduce yourself and your domain of expertise?
2. How do you define DevOps?
3. Which type of organizations / applications DevOps is best for?
   (a) And which is not?
   (b) Can you give me examples?
4. What is the added value that DevOps provide?
5. Why is added value important?
6. According to your role, how do you define the stages/steps of DevOps?

Section II: In this section, the questions will be focusing on the security aspects of software production.

1. Where does security take place in these mentioned steps?
2. How is the security testing is planned and executed?
   (a) Who is involved?
3. What types of tests / security tests are performed in your team?
4. How the results of tests are interpreted?
   (a) In which situations security test results can stop deployment?
   (b) Who decides on aborting deployment? Who decides if an increment passed the test, or not?
5. What is your team's strategy on software security?
   (a) How do you prioritize threats?
   (b) When do you review this strategy?
6. What is the role of the other teams (Development and Operations) in deciding these priorities?
   (a) Are they involved in the process of security policy?

Section III: This section aims to understand the role of security within the DevOps team.

1. Is the security team involved in deciding development requirements?
   (a) Yes? What is their tasks?
   (b) No? Why not?
2. How are security measures decided for an increment?

Section IV: This section is to get an overview about the main concepts of security in DevOps.

1. What is your understanding of the concept Security by Design?

2. What is your understanding of the concept Shifting Security Left?
3. What is you understanding of the concepts DevSecOps?

Thank you very much for this valuable information. I will transcribe this interview and further analyze its content. All the information will be anonymized. May I get back to you should I have further questions? Do you have any other questions or comments?
Thank you very much.
:: Stop Recording ::

Appendix C

Case Study Protocol

C.1 Background

The case study is designed to answer the sub-research questions 2 and 3. The RQ2 aims to identify the impact areas of implementing DevSecOps, while RQ3 aims to create a framework of DevSecOps. Scientific research presented in the literature review is used as a base of this case study and for the proposed framework.

C.2 Design

Following Yin's classification of case studies, this case study is designed as a single case with multiple analysis units. This design is selected because the case is the DevOps process itself rather than a specific team or teams. The units of analysis are Development, Operations and Security.

C.3 Data Collection

Data will be collected using 4 techniques:

1. Interviews.

2. Scientific Publications (Literature) and White Papers from companies.

3. Strategy documents.

4. Standards.

All the collected material will be classified in 3 main categories representing the analysis units.

C.4 Analysis

All data us analyzed using Nvivo, a qualitative research analysis tool. In this environment, all data is inserted and coded into three main nodes each representing one of the DevSecOps perspectives. Triangulation is used to draw conclusions.

C.5 Case Study Interview Protocol

The goal of this interview is to understand the main components of DevSecOps framework and the impact areas. The planned time for this interview is about 60 minutes. For the purpose of further analysis, I want to record this interview.
May I record this interview for further analysis? All information will be anonymized.
:: Start Recording ::
Just for the record, do you approve the recording of this interview?

C.5.1 Section I:

1. Would you please briefly introduce yourself and your domain of expertise?

2. How long have you been working in this field?

3. Do you define yourself as Development, Security or Operations?

4. How do you define DevOps? What does DevOps mean to you?

5. Where does security take place in the whole picture?

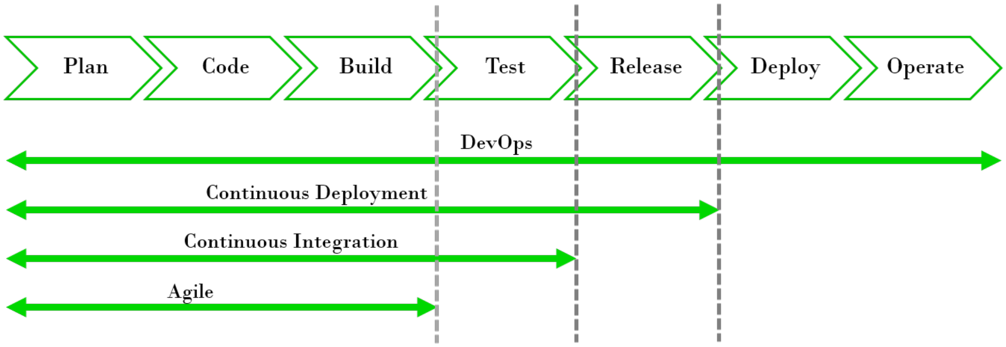


Figure C.1: DevOps Stages based on Rathod and Surve [83]

C.5.2 Section II:

1. Why do we need to secure applications? What happens if we don't include security?
2. Is the size of the application, or the data it is processing, important for determining the need for / type of security and how complex can this security be? If yes, what is an appropriate size?
3. How can an organization realize the value of DevSecOps? (i.e. customer trust, less security related issues in testing?)
4. How DevSecOps helps organizations adhere to security regulations? (e.g. GDPR)
5. How DevSecOps helps organizations comply to security standards? (e.g. ISO:27034)
6. How can DevSecOps impact the agility of software delivery?

C.5.3 Section III:

1. Starting with the DevOps concepts, what would change in the DevOps' team culture?
2. Is it necessary to add a team member who is responsible about security? What would be his tasks? If not, why?

3. Security has a blue team and a red team, each has a specific task. Should a DevSecOps team include any of them? Or both?

4. Why is automation important in security? What tools exist?

5. How can DevOps maturity impact the implementation of DevSecOps?

6. Looking into Figure C.1, we see the stages of DevOps. In a perfect DevOps implementation: What security measures would you include in:

   (a) Plan stage?

   (b) Code stage?

   (c) Build stage?

   (d) Test stage?

   (e) Release stage?

   (f) Deploy stage?

   (g) Operate stage?

7. For the validation of the results, there will be a focus group where you can validate my results, would you be interested in participating?

Thank you very much for this valuable information. I will transcribe this interview and further analyze its content. All the information will be anonymized.
May I get back to you should I have further questions?
Do you have any other questions or comments?
Thank you very much.
:: Stop Recording ::

Appendix D

Informed Consent

This interview is planned as part of my master thesis research in Utrecht University. I am doing my thesis in Accenture as an internship. The main topic of this thesis is DevSecOps and how security can be implemented in high speed teams. Through this interview, I am aiming to gain theoretical and practical knowledge about how security is handled in day-to-day activities and compare that to scientific research conducted in this area. The main goal is to develop a framework that structures DevSecOps.

The interview will be recorded, transcribed and analyzed to draw scientific conclusions. All of the information, people, companies and examples mentioned in the interview will be confidential and used only for scientific research. The interview will always be looked at in the context it is representing. The recording will be private, it will not be shared with other employees inside or outside Accenture nor other organizations. Entities mentioned in the interviews will be anonymized to ensure confidentiality. The recordings will be permanently deleted after the research is completed and the concluded results will be used in my thesis.

The interview does not aim to harm you nor your organization, therefore, you have all the right to stop the recording or the whole interview at any point if you feel uncomfortable to continue the interview. Participating in this interview is totally voluntary and only for supporting scientific purposes.

If you have read the above statement and agree with it, please sign below.

Interviewee
Name:

Date:
Place:
Signature:

Researcher
Name:
Date:
Place:
Signature: