UNIVERSITY OF TURKU

UNIVERSITY OF TWENTE.

A Model For Measuring Improvement Of Security In Continuous Integration pipelines

Metrics and Four-Axis Maturity Driven DevSecOps (MFAM)

Cyber Security (EIT Digital)
Master's Degree Programme in Information and Communication Technology
Department of Computing, Faculty of Technology
Master of Science in Technology Thesis

Author:
Akujobi Chukuwukamneleanya Akujobi

Supervisors:
Prof. Dr. Seppo Virtanen (University of Turku)
Prof. Antti Hakkala (University of Turku)
Maarten Vink (BiZZdesign)

October, 2021

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Cyber Security (EIT Digital)
Programme: Master's Degree Programme in Information and Communication Technology. (EIT Digital Double Masters Degree Program In Collaboration with University Of Twente.)
Author: Akujobi Joshua Chukwukamneleanya
Title: A Model For Measuring Improvement Of Security In Continuous Integration pipelines
Number of pages: 64 pages
Date: October, 2021

Abstract

This Thesis researches the effect of adding security tools into CI pipelines. The thesis is based on "security by design" within the software development cycle. The CI pipeline is based on a relatively new topic area DevSecOps. In the CI pipeline, there are several ways in which security can be improved, as well as research to back this up. However, there are not many types of research on the measurement of improvement within this field.

This thesis first investigates the literature on the topic of DevSecOps and Software Security. Following this is a systematic review of existing systems. It is then concluded that Dynamic application security testing (DAST) and Static application security testing (SAST) tools are added to the CI pipeline to further improve security by design. These improvements and additions are then measured with The Metrics and Four-axis Maturity Driven DevSecOps (MFAM) model developed to measure security improvements by design in the CI pipeline.

The implementation and validation of this model were done in two methods. Firstly, the metrics developed in the model are validated through the use of standards such as OWASP SAMM and ISO27001 2017. Secondly, it is checked to see if the model can be used in practice to measure improvements using a case study on the company BiZZdesign. For this contextual investigation, applications in BiZZdesign CI pipelines were measured. The measurement came in two phases. In phase one, the measurement of the state of existing systems at BiZZdesign is done, and in phase two, The addition of SonarCloud(SAST) and OWASP ZAP (DAST) is done. Finally, a comparison and evaluation of the two phases are made to estimate the level of improvement in the CI pipeline.

Several observations and key takeaways were then made towards the ending of the thesis. First, the quantification of how much a system improved when security tools were added was achieved. The MFAM model detected the changes in the maturity of security when security tools were added as it showed an increase of one or two levels in each axis of the MFAM model. However, factors like agility, productivity, and integration methods all came as points that can affect the addition of security as a part and body of the DevOps cycle. As much as security improvements were seen through the model, the scope is limited as the model is only used in one case study.

Keywords: DevOps, DevSecOps, Continuous Integration pipeline, Security Measurement, Maturity.

Table of contents

1 Introduction

This chapter gives a short introduction to the concept of software development within DevOps. After this is done, the specification of the research's scope, relevance, and overall structure is given.

1.1 This Research

Software development has evolved dramatically in the preceding 20 years. We have seen new technologies, development languages, computing power, and much more being improved and introduced over the years. One of the factors which have been a driving force to the development movement has been development methodologies. There has been a movement from Waterfall-driven development to Agile development by some developers. This movement comes in to tackle the market's needs effectively in a reasonable period and with a contingency to iterate continually as much as the need might be [1].

Additionally, DevOps was acquainted with bringing about an even faster deployment and development [2]. DevOps depends on the joint effort of two groups, Development and Operations, which permits two groups to convey better and cooperate more efficiently to tackle issues. Also, DevOps encourages automation to empower quicker operations and development patterns, permitting companies to deliver quality and value sooner to customers by doing several tests, leading to fewer errors.

Today, many operational and production models in software taxonomy are influenced by variables like the development environment, the market, production environment, and licensing. These variables, in turn, generate values that are used to enhance the user experience for their customers as well as for customer satisfaction. An example is using cloud-based software over client-based software because it is much simpler to roll out updates. In cloud-based software, running an update is as simple as installing the updates on the cloud, and these updates are immediately available to the customers to access. In the case of client-based software, the updates have to be manually and separately installed on the client's devices in some cases. One can also see this in the licensing model. In this case, the software is given as a *service* with the use of a subscription model. However, it is a *product* when the customer is rather granted access through a license. There are several other aspects of these classifications in the software taxonomy used to classify software. However, DevOps works to bring faster value to the customers irrespective of the final deployment environment, production environment, or licensing model. This means that customers have access to updates, new features, and fixes of their given application at the fastest release date possible.

Although this seems nice, security can come to be the primary concern. There are many threats in this area, for example, "Shared technology, shared dangers", "Malicious insiders", "Data breaches", and much more [3]. Developing dependable software that can reduce these threats involves thorough security assessments. In a DevOps environment, developers tend to think about "How fast can changes and updates be deployed to production environments automatically?". One common thing across DevOps teams is that several updates can be deployed daily, weekly, or fortnights. In the race to deployment, security issues in some areas might be overlooked, missed, or may not be taken care of because of the expected delivery time.

Security requires time and resources, which can make the process slow and, in turn, slowing down the DevOps cycle or not being taken into serious consideration to deliver in time. This puts the software and customer's data at risk from lots of threats. Usually, in DevOps, security tests happen at the testing phase when they are in place. These tests and other tests happen only at this phase. Nevertheless, DevOps has several phases with possibly different environments, and specific tests are needed at each phase.

It has always been ideal for including security as an integral part of the entire app life cycle. DevSecOps is about built-in security, not security that functions as a perimeter around apps and data [4]. It is rather seen as continuous security within the Software development life cycle (SDLC). We see Continuous security as:

> "Transforming security from being treated as just another nonfunctional 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" - Fitzgerald and Stol [5]

This research aims to help organisations understand the significance of adding security tools into their DevOps cycle (With a focus on the Continuous Integration pipeline). It is finding out how to measure the improvement that a security check brings and its effects on agility in the deployment process are what the research will cover in a limited scope, as is discussed in section 1.3.

1.2 The Problem Statement

The DevOps team works to bring faster software delivery by using iterations and increments. This rate is not likely to slow down, but instead, attempts are being made to make this faster. On the other hand, the security team works to ensure that software's are robust in security against threats while keeping in mind the Return on Security Investment (ROSI). They also ensure that there are temporary measures in place when the security fails. Both teams seem to be working against each other as one is looking for quick deployment, and another is looking for robustness which takes time. Many organisations and enterprises do not know if their security tools are working. Research by HELP NET SECURITY showed that 53% of enterprises have no idea if their security tools are working [6]. It becomes essential to understand the extent to which security can improve when security tools are added and possibly their effects on agility. There is also a problem of how security teams can implement more of a DevOps style of working, where security improvement is deployed and measured faster and integrated within the DevOps process.

Additionally, it is often difficult for organizations to know if adding a security tool improves their overall security at each phase. Simply embedding security tools in a DevOps development process is not enough. If not configured properly, it can be rendered useless. It then becomes vital to know if these tools improve the DevOps process's security and align with DevOps' principles. This goal helps an organization tell if these additions are resourceful to them or not. With these reasons given, we can see that security is indeed a significant concern in DevOps, and with this, we can now give the problem statement as:

In DevOps, Security is usually seen as an afterthought due to its effects on agility. Nevertheless, security checks need to be shifted more to the left to help improve the overall production complex. However, adding some of these checks is not favoured due to a lack of understanding of its usefulness. Therefore, it is essential to understand if these security tool additions improve DevOps security.

1.3 The Scope

DevOps, as limited as it may sound, is a broad field. It has to deal with many different platforms and tools being used to achieve the required goal. This also implies that several companies might be using different methods for the implementation of their DevOps. In the perspective of security, applications, tools, testing environment, and production environment influences the type of security used. For example, Standalone applications installed on a system that is not connected to an external network have lower threats if it is physically appropriately secured than a cloud-stored application.

To be able to define the scope of this thesis properly, we have to take a little dive into the "The International Organization for Standardization (ISO)" conceptual model named Open Systems Interconnection (OSI). The OSI model has seven layers, as can be seen in Figure 2. Each of these layers has its security. For example, the way the physical layer is secured might not be the same way the application layer is secured against attacks and threats. Looking into the security of all these layers is practically infeasible for this master thesis. Therefore, only the security of the application layer is covered. DevOps has to do with the application layer, as it deals with software or application development. However, as already mentioned, it is a broad spectrum as well. DevOps involves planning, building, testing and deploying, delivering, monitoring and logging, and finally gathering [8]. The process up to the testing and deployed phase can be seen as continuous integration (CI) and continuous deployment (CD). This thesis would be focusing on, but not limited to, the security of the planning, building, and testing phase, i.e. Continuous Integration. The aim is to answer this central research question, "How much will security be improved when adding security tools into CI pipelines?" using BizzDesign as a case study.

1.4 The Relevance

Many software companies today do have DevOps integrated into their work cycle. Whether it is fully automated or not, it is an aspect many companies try to employ to achieve agility and quality to customers. Bringing in security at an early stage helps to cut down the overhead time when the right things are done initially. Thus, aiming at integrating both DevOps and security to reach their goals. Today, it is referred to as "DevSecOp, DevOpsSec, rugged DevOps, SecOps, SecDevOps, and Secure DevOps". It is still a relatively new field and becoming more and more adopted by companies by the day. There are many considerations involved when a company is adopting DevSecOps, which leads to many implementation

options. However, how can a company honestly know if its implementation method of security is truly secure enough?

The scope of this research is with companies that have a CI pipeline. From research, there are many ways to try and improve software through the CI pipeline. Nevertheless, to the best of my knowledge and research, there is not enough scientific research on the amount or level of improvement in which additions of security tools to a CI pipeline brings to the SDLC. This thesis covers a research problem in design science within the scope of security. It aims to respond to an informative question about security by design in a field within DevOps in regard to software organizations.

1.5 The Research Question

The research aims to integrate security concepts in DevOps' development and operation phases, focusing on CI pipelines. Currently, security in DevOps is looked at in the application's testing stage and after its production, which depends on the organization's application. For example, an organization could have penetration testing in these two stages, and other tools or tests are used. However, research shows that security is not applied in all stages or phases in practice. CyberArk found that 94% of organizations had adopted DevOps, but only 28% had fully integrated security teams and processes throughout the application development process [7]. Security is critical but sometimes seen as a drag to innovation, making it to be actively avoided [7]. This brought about two aspects to be looked at. The first aspect is "continuous security". This aims to determine how security can be integrated, measured, and evaluated in each phase of DevOps with a concentration on CI pipelines. The second aspect is "Effects on Rapid value delivery". We need to know how this security affects the DevOps timeframe. This then leads to the main research question.

1.5.1 Main Research Question

RQ1: " How much will security be improved when adding security tools into a CI pipeline?"

1.5.2 Sub-Research Question

The "main research question" is divided into different parts, which we see as "Sub-Research questions." It comes in two main parts intending to answer the main research question at the end of the research.

- Sub-Research Question 1 (RQ1.1): **"How do we measure security improvements of a CI pipeline when evaluating security design tools or measures added to it?"**
  This sub-research question aims to discover how organizations can measure the level of security for added security measures and tools in a CI pipeline. Here, A metrics or model will be brought in place to check the level of security that the added security tool or measure provides.
- Sub-Research Question (RQ1.2): **"In practice, is it possible to measure security improvements when security tools are added to a CI pipeline?"**

Setting up a model or metrics for measuring the CI pipeline's security level is good, but how does this model work in practice? This sub-research question aims to validate the model by discovering how this model or metrics developed can be applied in practice. To be able to achieve this, a case study is required. In our case, this is the organization" BiZZdesign".

1.6 The Research method

The primary research method that is utilized for this thesis is Hevner's Design Science Research method [9]. Design Science Research is the design and investigation of artefacts in context, which is a solution-based method. For example, *"How to design an architecture to conform to investment goals."*. In our case, we look to see *"How much will security be improved when adding security tools into a CI pipeline?"*. This research method aims to provide real-life solutions to problems by providing iterative solutions related to the existing knowledge base and environment covering the problem. First, one carefully needs to study the problem environment, then, based on this, develop a solution that applies a fix to the problem. Finally, an evaluation of the artefact is done. Denning and Metcalfe beautifully defined it as a method that "seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished [10] "

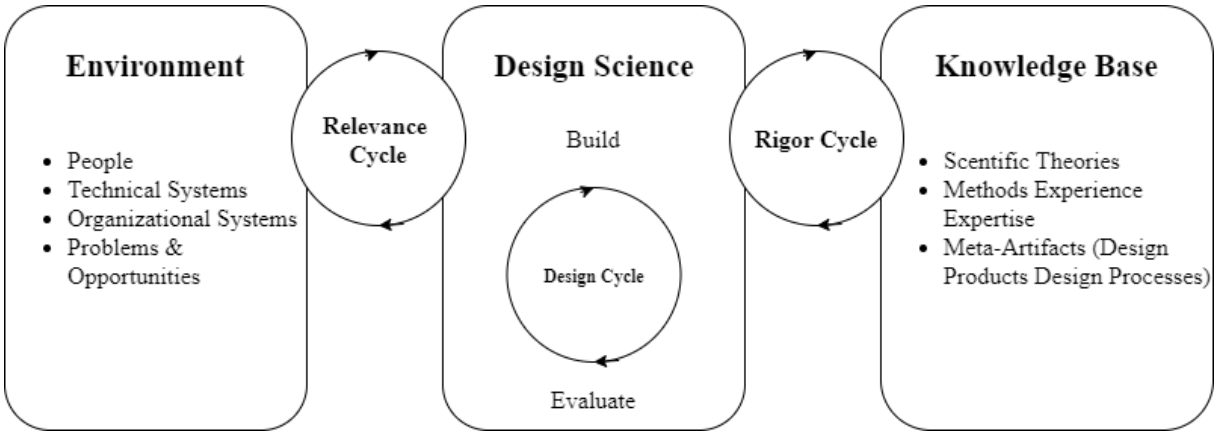

Figure 1: Hevner's Design Science Research framework [9]

Figure 1 illustration shows three inherent research cycles which connect three main domains to produce a validated result. First, the environment domain is where the problem and opportunity develop. It usually involves the people, organizational systems and technical systems. Second, the knowledge base domain is the reference point for all the information around the problem area. This information includes the logical exploration or scientific research, the experience of individuals who work in that particular problem area and current elective solutions. Finally, design Science is how to incorporate the information, create new information, tackle the issue, and validate or approve the proposed solution. It is an iterative interaction of building and validating how proposed arrangements fit in that circumstance.

1.6.1 The Relevance Cycle

One of the primary purposes of design science is to improve real-world problems through new innovative artefacts. The relevance cycle bridges the contextual environment of the research project with the design science activities [11]. The environment relates to the design science domain in two stages: the beginning and end stages. In the beginning stage, researchers gather requirements in which a context is built for the research. E.g. What are the problems and opportunities ?. The environment receives the research results at the ending stage and applies them to a real-life situation to solve the proposed problem.

1.6.2 The Rigor Cycle

The Rigor Cycle connects the design science activities with the knowledge base of scientific foundations, experience, and expertise that informs the research project. This connection is also in two stages [11]. In the first stage, researchers look for enough information or knowledge base to ground the research. This knowledge base contains state of the art research, expert opinion, existing artefacts, solutions, and processes. The second stage is when a solution is developed, and this information is used again where it is deemed applicable.

1.6.3 The Design Cycle

This is where the hard work of design science research happens [11]. The central Design Cycle iterates between the core activities of building and evaluating the design artefacts and processes of the research [11]. It combines the environment and the Knowledge base to bring about real solutions which are innovative and solve the problem context. It is well divided into two stages, building a solution or artefact and testing out this solution. The requirements of the environment and knowledge base are what make up the building stage. The testing stage, also seen as the validation stage, checks for the usefulness of the solution. This process or cycle keeps happening until a satisfactory result is reached.

1.7 Thesis Structure

This thesis will consist of seven main sections. In Section 2, the background literature of the topics within the thesis is covered. This section aims to give the reader a basic understanding of the topic area. In section 3, A systematic review of the existing systems is done. This section will cover a more detailed view of the topic area and how they currently work in practice. Section 4 covers the model developed to find a solution to RQ1.1. The next section being section 5, aims to answer RQ1.2. Finally, a conclusion is done in chapter 6 with key observations and takeaways.

2 Background

This chapter covers the background and literature concerning this thesis—these elaborate on some terms and topic areas that may not be known to the reader. The goal here is to give the reader an understanding of this preliminary research's terms and topics.

2.1 The Software Development Life Cycle (SDLC)

Software development is not new to many today. It is required by Computers systems to help solve real-life problems. When there is development, there is usual practice, planning, testing and evaluation in which there is repetition to enable progress. This same principle applies to software development. A software development life cycle is a methodology or framework used to plan, manage, and control the process of developing an information system [12]. The purpose of this is to help reduce risk, allow evaluation, allow teaming and most importantly, lead to good software. This methodology comes in different phases: planning, designing, development, testing, and deployment. These phases are illustrated in Figure 2.

2.1.1 SDLC phases

- **Planning:** In this phase, the project leaders or planners evaluate the project to be developed and plan based on this development process's structure. These may include costs, timetables, teams and leadership structure, and what they might see fit. An essential aspect of this phase is the definition of requirements. This requirement helps all parties know what the application will do or what they will do during the development cycle.

- **Design**: After a clear definition of what requirements are needed, the design phase then starts. This phase is where one gets an idea of how the software application will work, usually involving the user interface, architecture, platforms, communications, programming, and security. In some cases, the design comes as prototypes, and after several validations from client and project leaders can finally be accepted as a production design.

- **Develop:** This phase involves the actual program development, in which developers write the program in coding languages. Depending on the project's workload, this program writing could be done by a single developer or a team of developers. However, this is the phase that turns design into implementation.

- **Testing:** It is a critical aspect of the cycle as it aims to test that what has been implemented does what it is supposed to do. It is essential to do this test to ensure no bugs and errors during the development phase. These tests can be security tests, feature tests and much more.

- **Deploy**: After all required tests have passed successfully, the software is now ready to be released to the user. In this phase, the program is released into the production

environment. This environment is where a customer can then access the software for use.

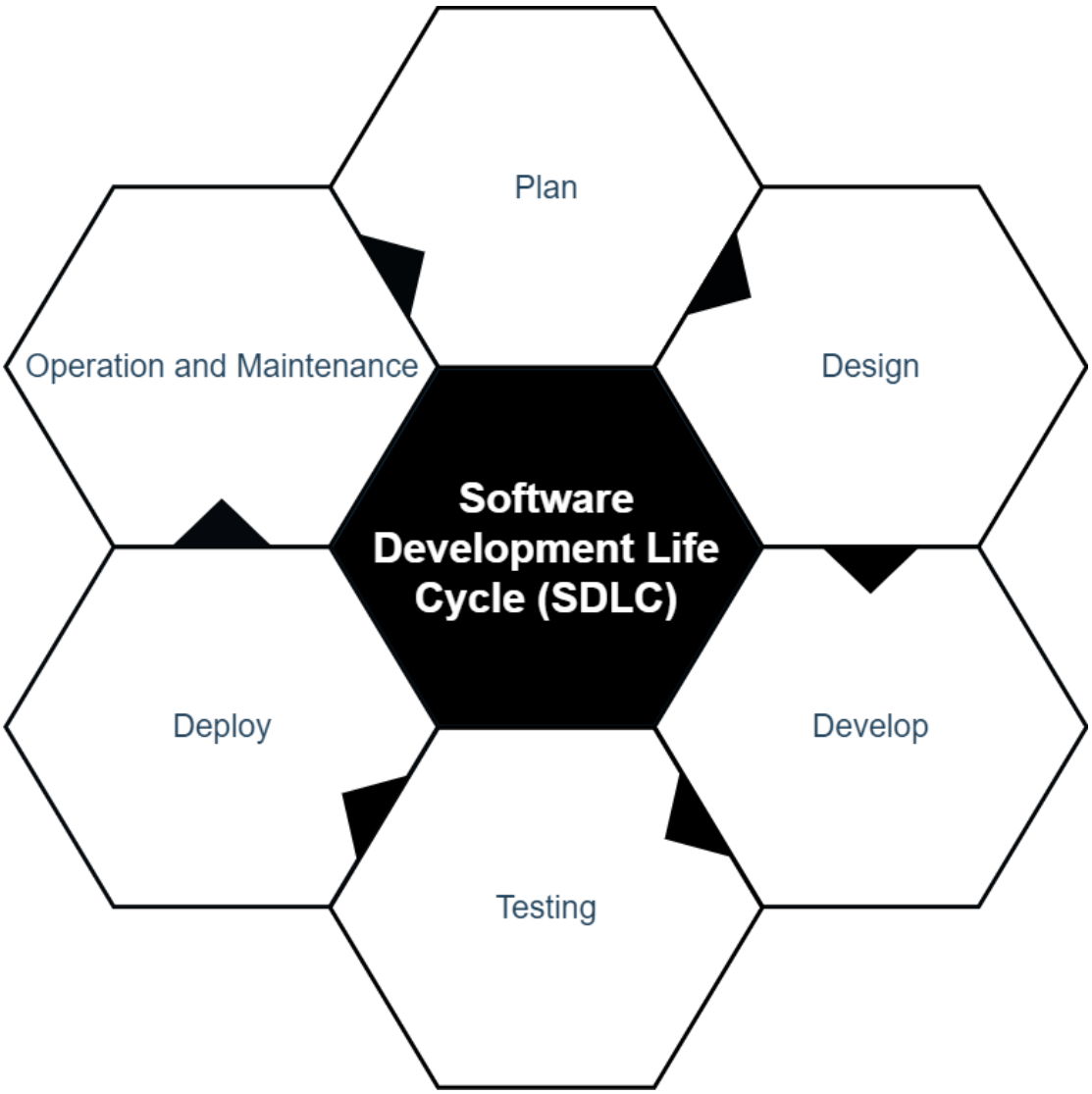

Figure 2: The Software Development Life Cycle (SDLC) Phases

- **Operations and maintenance:** At this phase, the customer should already have access to the software release. However, users find bugs that were not discovered in the testing phase. This aspect is where maintenance comes to play. The developers have to consider these bugs and then find the best fix for them.

2.2 The SDLC Methodology: DevOps

The SDLC has several methodologies or models in software development, in which each of them serves its purpose. Some of these methodologies are the waterfall model, agile model, iterative model, Spiral Model, DevOps model and more. This research focuses on the DevOps model and, therefore, only limits the literature to this.

2.2.1 What Is DevOps?

IT operations is a vast department with steady growth over the last few decades. It is generally divided into two aspects. The 'operations or Ops' aspect goal is to maintain a steady working IT infrastructure to support the business. They are usually risk-averse, which means that procedures typically prohibit actions that would increase risk. The 'development or Dev' aspect goal is to develop innovative solutions rapidly that answer to a changing business environment. This "Dev" aspect characteristically embraces risk [13]. This is because there is the development of new services or features in which risks are inevitable. It typically shows that there is isolation between the two. DevOps is a model that recognizes that both cultures need to be integrated, bringing about the same shared goal and breaking the barrier between these two aspects.

"DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support." - Carlos Gomez [14]

2.2.2 DevOps in Practice

In section 2.1.1, we saw the typical phases or stages of the SDLC in which each methodology under the SDLC commonly has. However, its application of these phases is different. For example, the way the waterfall methodology works is not the same way the DevOps methodology works. In DevOps, it is typically divided into eight phases, as shown in Figure 3. However, this phase still correlates with the phases of the SDLC with a bit more depth in some areas. The correlation can be seen in table 1.

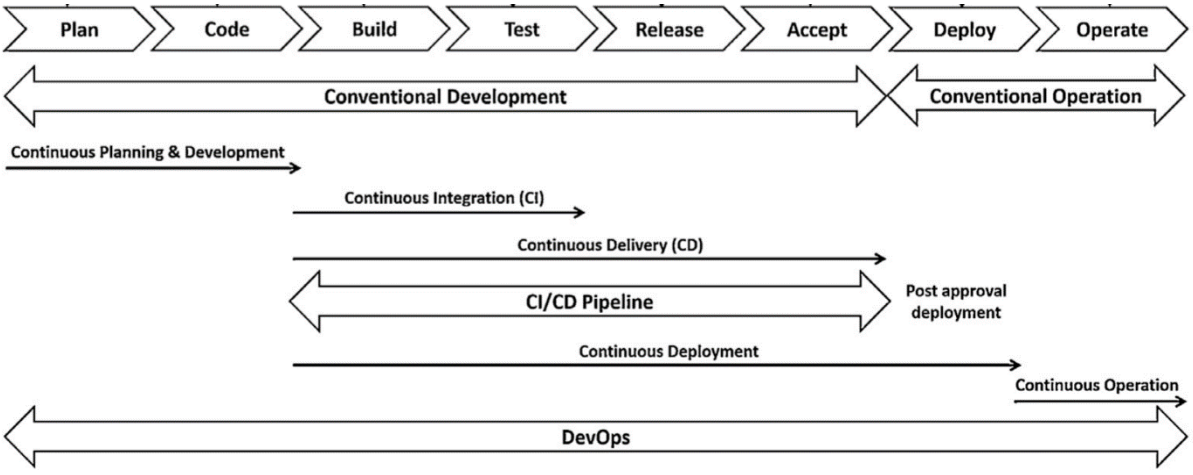

Figure 3: DevOps In Practice [16]

SDLC	**DevOps Life cycle**
Plan	Plan
Design	Code
Develop	Build

Test	Test
Deploy	Release, Accept, Deploy
Operations and Maintenance	Operation

Table 1: Correlation of DevOps Phases with SDLC Phases

These phases are, in practice, categorized into several parts, as can be seen in Figure 3.

- **Continuous planning and development:** This category includes the planning and coding phases of the DevOps life cycle. These phases might be iterative, i.e. they might be version control mechanisms in place.
- **Continuous integration (CI):** Simply put, it is a standard development practice in software engineering organizations where teams or their members integrate and merge developed works or codes frequently, i.e. this could be per day, week, fortnight or month. It is also a typically automatically triggered process comprising inter-connected steps such as compiling code, running unit and acceptance tests, validating code coverage, checking coding standard compliance and building deployment packages [5]. This continuous integration helps to bring faster time to production for developers, which benefits the organization and customers.
- **Continuous delivery:** Continuous delivery is the practice of continuously deploying good software builds automatically to some environments but not necessarily to actual users [5]. It means that the goal here is to ensure that an application (after passing the automated tests and quality checks successfully) is always at a production-ready state. From Figure 4, we can see that to have continuous delivery, one should have continuous integration.
- **Continuous deployment (CD):** Continuous deployment implies continuous delivery and is the practice of ensuring that the software is continuously ready for release and deployed to actual customers [5]. Sometimes it is often debated if there is any difference between continuous delivery and continuous deployment. The answer to the question is "Yes". The difference is simply the production environment, i.e. the actual customer. These relationships between the CI, CD and continuous delivery can be seen in Figure 4.
- **Continuous logging and monitoring:** This practice involves ongoing monitoring of both the code in operation and the underlying infrastructure that supports it [15].

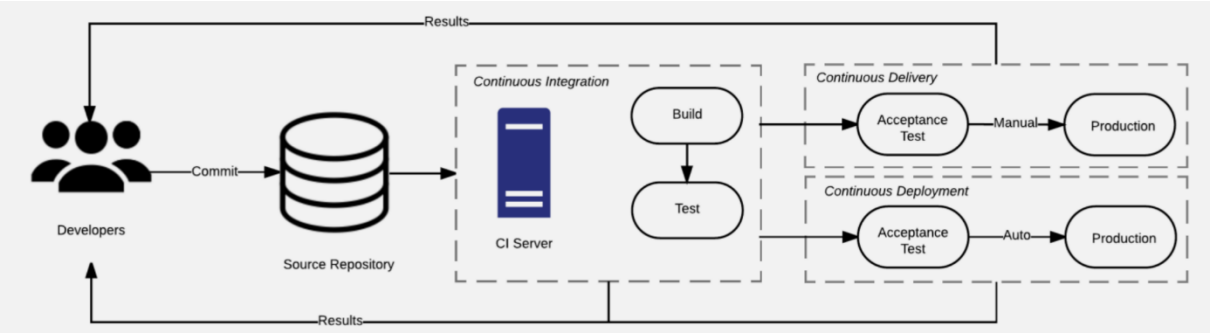

Figure 4: The relationship between continuous integration, delivery and deployment [2]

2.3 Software security

Software security is about making software behave correctly in the presence of a malicious attack, even though software failures usually happen spontaneously in the real world—that is, without intentional mischief [17].

Simply put, software production to perform the task it is intended to do is not good enough. Security is vital today, as cyber threats and risks keep increasing every single day. This point is also an important aspect when an organization is selecting a software vendor. In software security, there are some security principles based on requirements that a software should have, i.e. confidentiality, integrity, availability, authentication, authorization and non-repudiation [18]. These security principles are the underlying principles for software security. In table 2, We can see a layout definition of these software principles. Implementing these principles is not easy in practice, but it is a crucial software production requirement. For example, the security breaches or attack types might differ in different systems. It is as well possible for zero-day attacks. A zero-day exploit is a cyber-attack that occurs on the same day a weakness is discovered in software [19]. These weaknesses or weak points in which an attacker may exploit are known to be “Vulnerabilities". Therefore, security should not be an afterthought but rather be part of the foundation of a software system.

Term	**Description**	**Attack type**
Confidentiality	“property that information is not made available or disclosed to unauthorized individuals, entities, or processes.”	Password Attacks, Phishing and Pharming
Integrity	“property of accuracy and completeness.”	Man-in-the-middle attacks, Session hijacking attacks

Availability	"property of being accessible and usable upon demand by an authorized entity."	DDoS (Distributed Denial of Service attacks), Electrical power attacks
Authorization	"Property of access privileges being granted to users, processes or programs."	Credential/Session Prediction, Session Fixation
Authentication	"Property of establishing the validity of a message, transmission, or sender to be able to verify the user's authorization."	SQL injection, Brute Forcing
Non-repudiation	"Property of non-denial of participation by personals in a transaction at a later time after participations."	Web Parameter Tampering

Table 2: Software Security Principles [5],[20],[21]

2.3.1 A Call For DevSecOps

DevOps has been looked down on for its lack of attention to security. This reservation comes from DevOps' main goal: rapid time to release software products or features. This rapid delivery raises the question of the prioritization of security in the development process. However, agility does not mean a lack of efficiency. It is also essential to know that security measurement is not easy. Security is usually seen when there is a lack of it. This statement means that a system without security is more likely to bring headlines when there is a cyber-attack than a fortified system in which it could go without any notice.

Additionally, the IT persons and the security experts' level of collaboration is not easy to evaluate. About 64% of administrative management view that security teams are involved in technology design and deployment, where 39% of software developers accept this reasoning [22]. When there are conflicting views on the collaboration level and what is being done, it becomes difficult to know the security state in that organisation's development process.

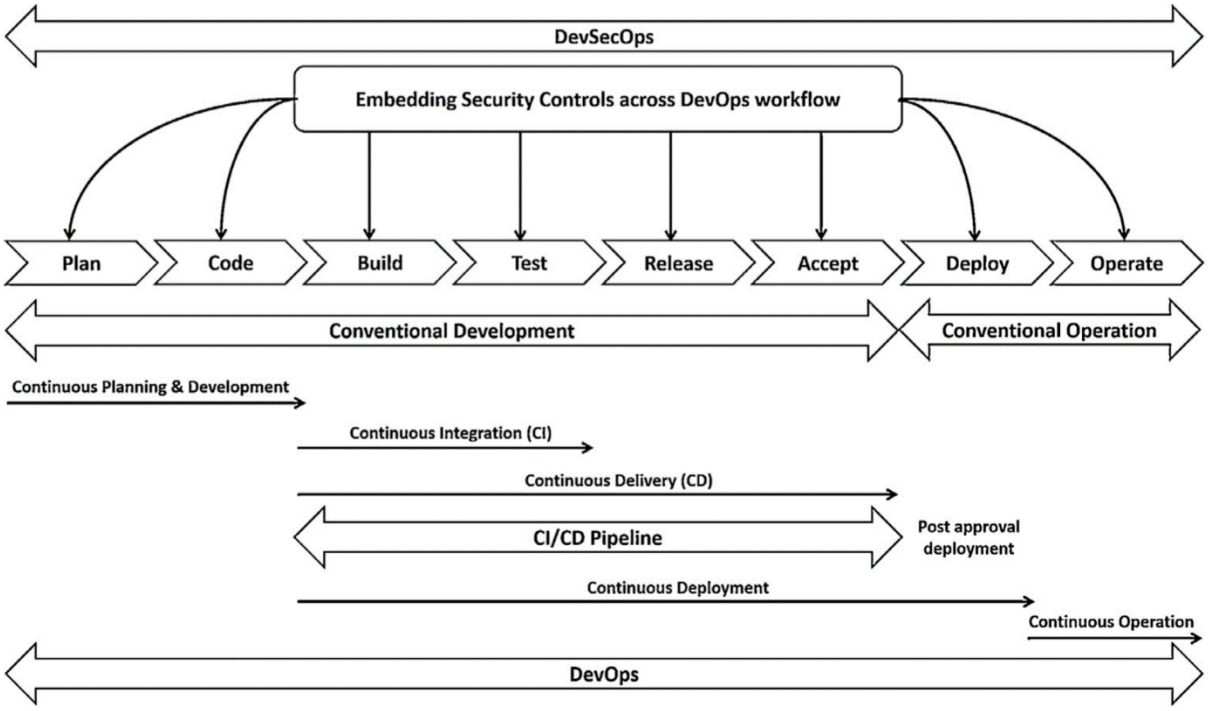

Figure 5: DevOps with security (DevSecOps) [16]

DevSecOps (development, security, and operations) automates the integration of security at every phase of the software development life cycle, from initial design through

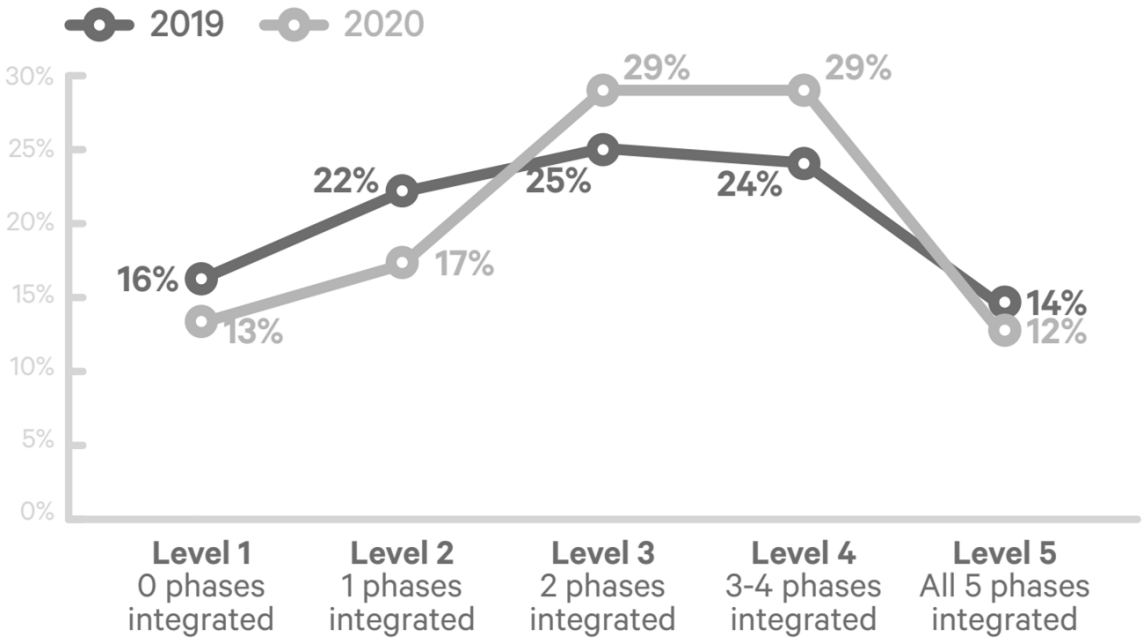

Figure 6: Devops Level of Security Integration 2019 and 2020 [24]

integration, testing, deployment, and software delivery [23]. Figure 5 shows that security is not an afterthought but rather a continuous development process. However, this is not the case in many organizations today. Research from Puppets (2020) shows that up to 13% of organizations do not have security integrated into the DevOps phases [24]. It also showed that 29% of organizations integrate security in 2 to 3 phases, while 12% have security integrated into their DevOps phases [24]. These readings are illustrated in Figure 6. These opinions on collaboration levels and what happens in practice differ; it becomes difficult to know the state of security in an organization's development process. All this shows that security is not so easy to integrate into DevOps.

2.3.2 Why is DevSecOps difficult in practice?

Organizations often insert security at some phases in DevOps, as shown in Figure 6, but we have seen that it is required at all phases of the DevOps life cycle. So the big question here might be, “why is it difficult to integrate security?”. One cynical yet not uncommon view on this is that security practices are a bit more than a security theatre, i.e. which is made to avoid responsibility or blame, rather than improve the security posture measurably [24]. One other reason is that security is seen as a hindrance to development due to the delays security checks impose on the lifecycle. It is usually added at the end of the delivery lifecycle, which means when these security checks find some issue, it then imposes unexpected work on the DevOps teams. This incredible work or unplanned work then brings about delays and further frustrations.

3 A Systematic Review of Existing Approaches

This section is split into two parts. In the first part, the preliminaries of how a standard CI pipeline works is explained. In this case, we show BiZZdesign architecture as an example. The second part is a systematic review of the existing approaches to inserting security checks within the CI pipeline.

3.1 Example Existing CI pipelines

In the Background, An explanation of how a general DevOps system works was explained. In addition, we saw how each DevOps phase relates to bringing about a final software product release. However, We also need to see how these phases work in practice with an existing system. Therefore, these preliminaries help the reader understand how an existing DevOps system is approached in practice. For this, we take BiZZdesign as our example system.

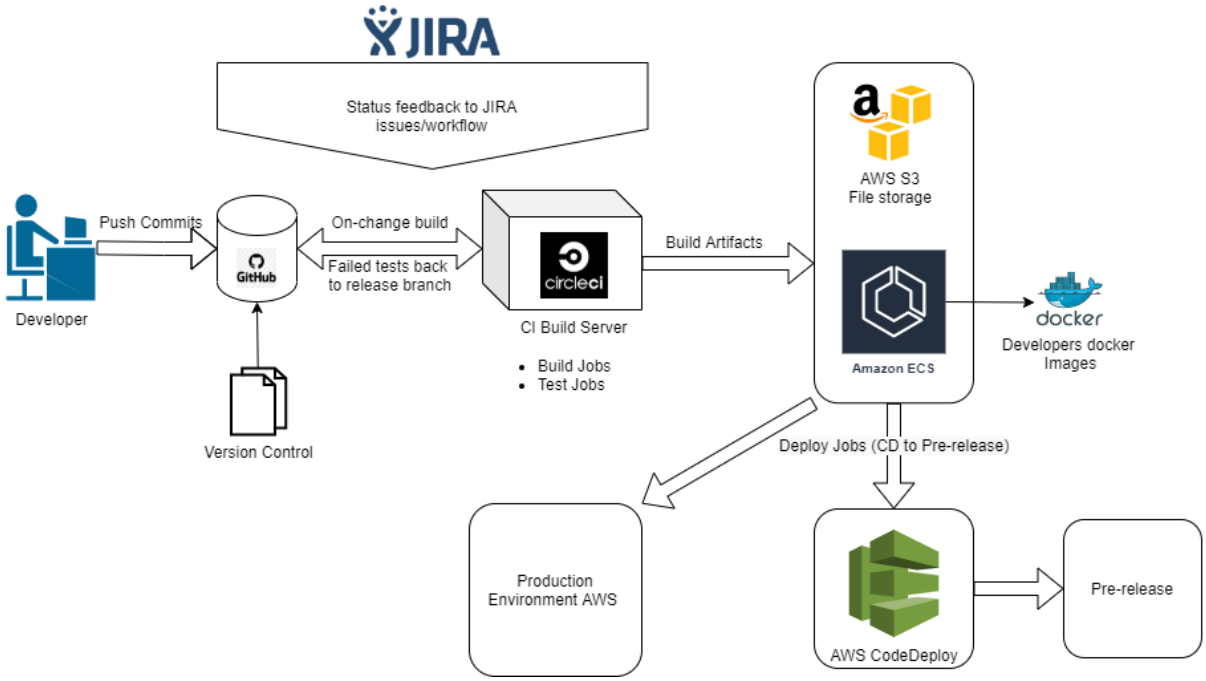

Figure 7: BiZZdesign's Pipeline Architecture

An example of an existing DevOps pipeline is seen in Figure 7. Due to a non-disclosure agreement, in-depth coverage of how exactly this architecture works is not covered. However, A relation between the architecture and how it applies to each DevOps phase is more critical. One can see the relationship in table 3.

CI/CD Phases	Relation with example pipeline
Code	Developers Push code to Github
Build	GitHub to CircleCI
Test	Circle CI
Release	Build Artifacts to s3/ECS
Accept & Deploy	Deploy to pre-release or production environment

Table 3: Relationship of CI phases with example architecture

3.2 Security tool Options for CI phases pipelines

There are many tools used to add security to CI pipelines. However, this thesis focuses on measuring the security improvement a tool brings in CI pipeline phases rather than the type of tool. One example can be seen in Figure 8, an overview from Microsoft on integrating security into a CI pipeline. These options come as security validation methods, intending to integrate the security in the CI pipeline. This section brings an understanding of the options of adding security tools within a CI pipeline.

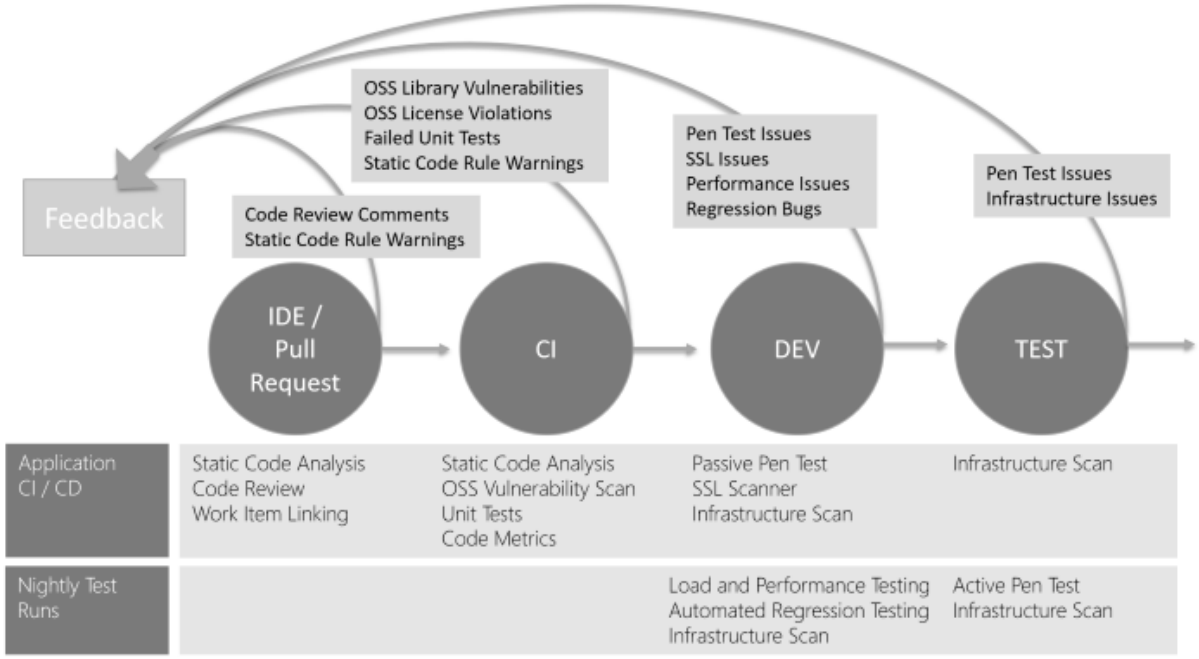

Figure 8: Security options added to the development process by Microsoft [25]

Finding security tools for a CI pipeline can be daunting. This is because many tools from different vendors and similar features can be added to test security. Covering them all is possible but can as well be redundant. First, To get the best out of what matters, the categorization of these security tools based on their validation method is done. Following this, the application of where this tool is used in practice is evaluated. Based on the security validation method, we look at the following: static application security testing (SAST), open-source security vulnerabilities (OSS), dynamic application security testing (DAST), Unit Testing and finally, Interactive Application Security Testing (IAST). Integration testing is one other aspect. However, it is not covered in the scope of this thesis, as it's considered low in security relevance. Furthermore, it has more relationship with infrastructure security which is not part of the scope of this thesis.

3.2.1 Static application security testing (SAST)

Static application security testing (SAST) checks for security weaknesses in the CI pipelines. Its been around for a while now and works based on the concept of white-box testing. A SAST tool also works by analysing given codes in a non-runtime environment for bugs and paths that lead to security

weaknesses or risks. It allows developers to find security vulnerabilities in the application source code earlier in the software development life cycle [26]. This also ensures conformance to coding standards before codes are executed. Due to its vulnerability findings at an early stage, developers can resolve security issues before they cause real damage. SAST tools are therefore used in IDE or pipelines by developers. It can as well used to improve the security of infrastructure as a code (IaC).

3.2.1.1 Drawbacks of SAST

As much as it is easy to deploy SAST tools, SASTs throws a lot of false positives. Hdiv showed from a test of the speed, coverage and accuracy with the OWASP Benchmark that the best SAST tools have about 53% false-positive [27]. These false positives are because SAST does not consider the presence of other security countermeasures and lacks visibility during runtime [27]. This situation makes it challenging to balance not showing false positives and showing all available errors. Usually, SAST tools will like to scan for all possible errors as they would like to give all potential security flaws. However, this comes at the cost of more false positives. One critical problem of many false positives is that developers then take the tools less seriously. This is due to the headache and time it would take to go through the false positives to find what is essential to fix. There is also the drawback of false-negative where vulnerabilities result, but the tool does not report them [28].

3.2.1.2 SAST Tools Security Options

Some many tools or options attempt to apply SAST methods. Some of these tools are in a list made by OWASP on SAST solutions [29]. Some popular mentions that might best fit the case study are CodeClimate, SonarQube, Checkmarx, and Puma Scan. Later in the thesis, we see which tools are taken to test security improvements in a phase.

3.2.2 Dynamic application security testing (DAST)

DAST is a form of black-box testing typically used to find security weaknesses and vulnerabilities in an application at runtime. It does that by employing fault injection techniques on an app, such as feeding malicious data to the software to identify common security vulnerabilities such as cross-site scripting and much more [26]. This method is different from the SAST, which is instead a white-box form of testing. SAST does not work in the application's runtime but rather with the source code provided to it. DAST tools can be added to SAST tools to add more robustness to the security testing package.

3.2.2.1 Drawbacks of DAST

DAST is not going to flag coding errors, at least not down to the code line number [43]. It looks at applications from the exterior and determines the presence of risks by looking at the server's response to a battery of tests, meaning DAST has no visibility of the application's internal workings [27]. This means that not all vulnerabilities could be discovered, as there is no full coverage of the application. It can be expensive and difficult to automate since DAST Operation needs to be done by an experienced AppSec team, like penetration testing. However, the main issue with DAST is that it can take around 5 to 7 days to scan [27].

3.2.2.2 DAST Tools Security Options

Like with SAST, many tools can perform DAST, which some of these tools can be seen on OWASP [30]. They are not all the same and have their own method and style of achieving DAST. The majority are commercial, but some popular mentions are OWASP Zed Attack Proxy (ZAP), Netsparker, Checkmarx and Acunetix. Note that none of the tools mentioned here is endorsed but rather put by the popular view.

3.2.3 Interactive application security testing (IAST)

IAST is relatively new to the world of application security checks. It is a bit similar to SAST and DAST, yet different, as it focuses on detecting security issues in the code of applications by combining both SAST and DAST approaches. IAST, like DAST, functions dynamically to identify bugs and problems on an external scale. It does, however, run as an agent within the application server, evaluating codes like SAST. This allows it to see a potential flaw more clearly and determine if it is really a true positive. As a result, the number of false positives and false negatives is reduced. Simply put, IAST works in real-time by analyzing applications for security issues within its applications' traffic and execution flow.

3.2.3.1 Drawbacks of IAST

The IAST architecture is based on code instrumentation, and therefore it is language-specific in terms of server-side infrastructure [27]. IAST lacks coverage across specific languages and only supports modern technology frameworks [31]. Another drawback of IAST is that it requires a mature test environment. This is because it does not work so well with the older software development environments and architecture [31]. Furthermore, one needs to consider the fact that it is not yet widely adopted [31].

3.2.3.2 IAST Tools Security Options

As IAST is relatively new, there are few security tools in this area that provide this solution. OWASP [32] could find only one to be open source, while the others are rather commercial. Some popular mentions are Synopsys and Contrast Community Edition (CE). . Note that none of the tools mentioned here is endorsed but rather put by the popular view.

3.2.4 Open Source Software (OSS) Security Tools

OSS is used today by an enormous number of software developers. A study by Paula Rooney showed that more than 50% of the Global 500 use vulnerable open source components [33]. Furthermore, about 43 per cent of developers say that OSS provides better competitive features than the commercial equivalent [34]. These researches show that OSS is here to stay, but nothing is perfect in this world, which also applies to OSS. OSS refers to the open-source libraries or components that application developers leverage to develop new applications and quickly add features to existing apps [30]. It is also referred to as software composition analysis (SCA) by Gartner, another aspect of static analysis [30]. The main difference between SAST

and SCA is that SCA looks into vulnerable dependencies while SAST looks into vulnerable codes
OSS cannot be trusted blindly, so one needs an OSS vulnerability checker or tool during the development cycle. This OSS security tool works by scanning all dependencies of packages available to it from the application. One example is checking the npm packages against versions known in its database to be vulnerable. In the case of a vulnerability, the tool makes awareness of this in terms of feedback or security alert.

3.2.4.1 Drawbacks of OSS security tools

One downside of using OSS security tools is the purpose of which it is made for security. As it is open-source, hackers can make manipulations in code or scanners to skip, for example, a known vulnerability during a scan. One other major issue is the fact that support might be unreliable. In the case of problems with tools, the support needed by a team might not be readily available, like in the case of a commercial environment. Another issue is false positives. Just like the reason seen in SAST, false positives could be a problem. For example, a library or dependency is used that is known to have a vulnerability. However, the vulnerable part is never used in the code. It is also a factor to consider when determining the quality of an OSS tool.

3.2.4.2 OSS or SCA security tools Options

OWASP recommends two ways of avoiding using OSS components with known vulnerabilities. One way mentioned is to keep the libraries updated, as security issues could be fixed in new updates. The way is to detect known vulnerable components. For this within a CI pipeline, OWASP divided this into two aspects.

- Dependency Analysis: The Dependency Scanning feature automatically finds security vulnerabilities in dependencies while developing and testing applications [35]. These dependency scannings can help in finding open source libraries with vulnerabilities. Some popular open-source tools mentioned by OWASP for this are Gymnasium, OWASP Dependency-Check and track, and Github Security alerts for vulnerable dependencies. Some commercial tools that could provide this type are Snyk, WhiteSource and Source clear.
- Container Scanning: This class of tools will scan the contents of a container, looking for issues before they are used as a building block for the application and a final set of checks before a container is deployed to production [36]. Some popular mentions by sang.org on these tools are Clair and Trivy.

3.2.5 Unit Testing

Unit testing is a test executed by the developer in a laboratory environment that should demonstrate that the program meets the requirements set in the design specification [37]. It is an aspect that can be important to security because it ensures that every code path gives the correct result and nothing more. In Java, for example, the unit is usually a class. Unit tests invoke one or more methods from a class to produce observable results that are verified automatically. In the case of no unit testing, it could be challenging to fix issues or bugs because

one might not know its effects on functionality and security. It could lead to functionality breaks which could then lead to a security-related issue.

Usually, companies should have policies for their unit testing regarding correctness and completeness in a review process. However, due to human error, the coverage might be limited in completeness, which is why unit testing tools are needed. These tools could cover all code paths to ensure that they are tested in some particular pattern. However, the major downside of these tools regarding security is that they cannot tell if the testing method is secure. For this reason, we will not be covering this area in the scope of this thesis.

3.3 Summary

We have seen so far several Security tool Options for the CI phases pipeline. A summarised overview of these options and the phases they apply to in the CI pipeline phases is shown in table 4. SAST only takes effect at the coding & build phase because it does not work at runtime. The testing phase usually is where functionalities are tested, and any reports on issues are sent back for corrections before release. DAST, IAST, and unit testing usually come at this stage, as some of its tools do their checks at runtime within these phases. Finally, Only IAST and DAST work at the smoke test or pre-release before being released to production.

Phase / Tool Set	**Coding & Build**	**Testing**	**Release & Accept**	**Deploy & Production**
	Continuous Integration		**Continuous Deployment**	
Static application security testing (SAST)	x	x		
Dynamic application security testing (DAST)		x	x	x
Interactive Application Security Testing (IAST)	x	x	x	x
Unit Testing	x	x		
Open-source security vulnerabilities (OSS)	x	x		

Table 4: Summary of security options in CI pipeline

4 Model For Measuring Improvement Of Security In CI pipelines

In the previous chapter, We saw the security options or checks added to a CI pipeline. This chapter looks at a solution that measures the security improvements these checks or tools add to a system. Therefore, this chapter aims to answer RQ1.1, *"How do we measure security improvements of a CI pipeline when evaluating security design tools or measures added to it?"*

Developing models is not as easy as many may think. It requires time, precision, testing and evaluation. To create a viable solution, we have to look back to the problem statement. Looking at the problem, one can see that the keyword is an improvement. This led to the research questions on the measurement of security improvement in a CI. One cannot show improvement in a system without quantifying it with measurements over a timeframe. A straightforward model of showing improvement in a system is with Goal question metrics (GQM). It works by setting a goal and then asking questions that can help check if the goal is reached. Then metrics are used or defined to help answer the questions. Finally, an analysis is done to determine if there are indeed improvements. Although GQM shows improvements, it does not show the level or amount to which improvement has occurred or instead, it is not easy to quantify how much a system has improved. This problem gave birth to "The Metrics and four-axis maturity driven DevSecOps (MFAM)" model. This model is based on a standard maturity model as well as questions to match these maturity metrics. In the following section, we elaborate more on this.

4.1 The Metrics and four-axis maturity driven DevSecOps (MFAM)

We will first explain the architectural construct of MFAM. To best elaborate this architecture, a diagram is created, as shown in Figure 9. In this architecture, specific attributes will be needed to be filled in to develop this model successfully. This architecture has five main aspects, which are as follows:

- **The Improvement Objective**: The improvement objective is an aspect or subject area in which improvement is required. For example, We want to check for improvements in the CI area within DevSecOps. So, in this case, the improvement objective is the CI pipeline.
- **Information Axis**: The information axis comes directly after an improvement objective has been determined. The goal of the information axis is to show the areas within that subject topic covered when looking into its improvement. For example, an improvement objective for a CI pipeline might want to look at areas like Static Depth or dynamic Depth or related. These areas are essential axis in which one has to look into when considering CI improvements.
- **Measurable concept**: The measurable concept comes immediately after the axis. This area is the concept within an information axis that we want to measure. This definition means that this aspect should be possible to measure. For example, under an information

axis such as Dynamic analysis, a measurable concept could be Static or Dynamic Depth for applications.

- **Questions**: To best understand what can be measured within a measurable concept, questions need to be asked. In the model, the questions are divided into two areas. One is the **maturity question**. The maturity question is the leading question that must be answered about a measurable concept. An example under Dynamic Depth for

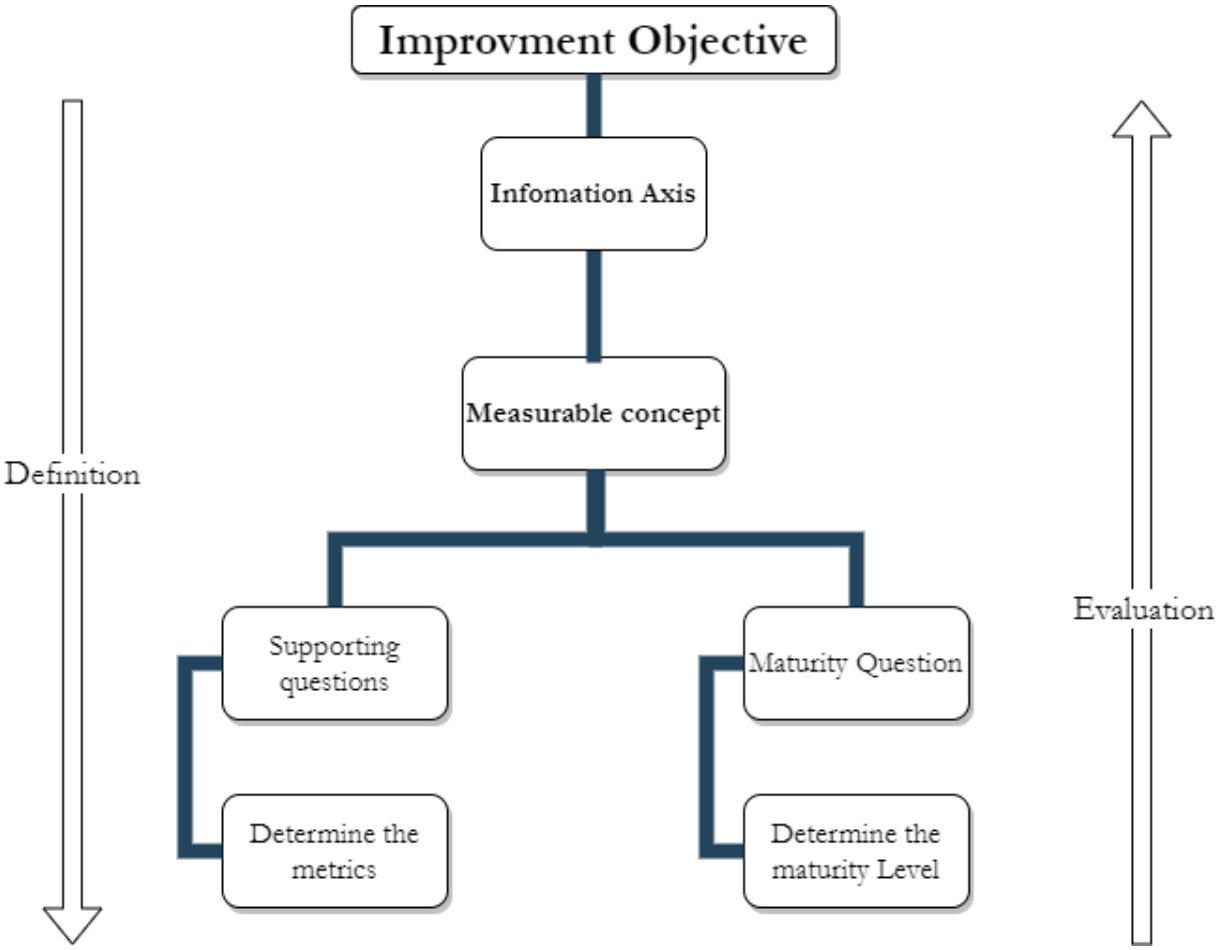

Figure 9: Model architecture

applications could be *"How deep are dynamic tests being executed?"*. The second area is the **supporting questions**. These questions help one understand better the measurements of improvement. However, they might not indicate the level or amount to which the improvement may have occurred. For example, a question might be, *"Does security scanning detect security problems like vulnerability?"*.

- **Metrics:** The metrics are measurements that are done to understand better the state of security. They are two categories for the metrics in this model. One is the maturity metrics, and the other is the supporting metrics. These metrics then help answer the questions under their respective question category, i.e. supporting questions or maturity questions. So, for example, one could have a maturity indicator ranging from one to five, which helps answer the question of maturity.

As can be seen above, these five aspects have been defined to make the MFAM model simplistic.

4.1.1 MFAM Attributes

To best elaborate on how this model works, a template is developed. In this template, specific attributes will be needed to be filled in to develop this model successfully. These attributes will be first described in tables 5, the areas of measurements will be covered in tables 7 to 11. This template is as follows:

Attributes	Description
Information Axis	The axis is the principal or central category in which the development of a security check is done. E.g. Static Depth.
Measurable concept	The measurable concept is a subcategory of an axis in which is measured for a specific environment. E.g. Static Depth for application
Measurable concept Description	This aims at providing information on what the measure concept is for.
Risk Description	It is crucial also to understand the risk involved if this check or tool is not in place. The risk description describes what risk is involved when this measurable concept is absent.
Required maturity level based on inherent risk estimation	This is the maturity metric that must be met to have a basic level of security within a CI
Questions	These questions help us understand the measurable concept and how they can be measured
Supporting/secondary Metrics	This area comes as an additional metric that helps us understand better if there was an improvement.
Primary metrics (Maturity Indication Level)	This is the primary indicator that shows the maturity indication of an axis.
Actual maturity level indicator	Here, we want the maturity indication level after the evaluation of a system.

Table 5: MFAM Attributes

4.2 The MFAM Five aspects Detailed (From Improvement Objective to Metrics)

Now that the template is set. The next step is to detail the values of the attributes of this MFAM model. To do this, we will follow the architecture as shown in Figure 9. First, we will set our improvement objective then set the information axis it covers. The next step is to set the measurable concept, questions (maturity question and supporting questions), and finally, the metrics (primary and secondary metric). The following process is called the **Definition phase**.

The **Evaluation Phase** is the opposite of the definition phase. In this phase, we instead work our way up to understand the meaning of the data gathered.

4.2.1 What is the Improvement Objective?

The improvement objective is already set through the research topic and question. The aim is to know how security by design can be improved in a CI pipeline when security tools are added. This improvement objective can be met if the improvement when these tools are added can be measured (Thus, the RQ1.2).

4.2.2 Information Axis Leads to Measurable concepts.

We have seen what the Improvement Objective is. The next step is to determine what information axis are involved with the objective. Table 6 illustrates how an information axis leads to a measurable concept. Each axis is then described, and the risk of what might happen if this axis is not checked is described as well. The measurable concepts defined are dynamic depth for applications, static depth for applications, intensity and consolidation. As the focus group is the CI pipeline, the measurable concept is limited to only the application areas within the CI pipeline phases. This means measurable concepts like dynamic and static depth for infrastructure are not included as infrastructure will also need to include the CD pipeline.

Information Axis	Measurable concept	Axis description	Risk description
Dynamic Depth	**Dynamic depth for application**	A dynamic depth is how detailed a dynamic security tool can be to find security-related bugs at runtime. This answers How profound are dynamic depths executed inside a Security DevOps CI pipeline?. Of course, this means one needs to know where these dynamic security tests are applied as well.	The absence of depth in dynamic security testing or checks leads to the risk of possibly seeing only security-related issues at the public attack surface. This is because tools are just thrown at the CI chain to see what it generates without requirements for authentication of the scanner with the target.
Static Depth	**Static depth for application**	A static depth is how detailed a static security testing tool can be to find security-related bugs before runtime. This brings the knowledge of the depth of static code analysis, which is expected in a CI Security DevOps chain. It is then paramount to know where	The absence of static depth in security testing or checks could easily cause vulnerable code to slip into production. These vulnerable codes could lead to actions from attackers who are keen on exploiting these vulnerabilities.

		these static security tests are applied.	
Intensity	**Intensity**	The level or magnitude of the performed attacks and checks done within a security DevOps CI pipeline. So here we want to know How intense are tests being executed inside a Security DevOps CI pipeline. So what is being checked?.	The absence of intensity causes one not to know certain levels of attacks that can be executed within the CI DevOps security chain. In this case, when specific scans are not done within certain timeframes or high configurations, it leads to unexpected attacks and undiscovered vulnerabilities.
Consolidation	**Consolidation**	Consolidation is an aspect that helps us understand how the results or findings from the security DevOps CI are used.	The absence of consolidation leads to a wrong interpretation of results and finding the correctness of the results. If the results are not understood or used correctly, the whole essence of the security check is pointless. Of course, this can then lead to attacks as vulnerability fixes might not be done correctly.

Table 6: Information Axis Leads to Measurable concepts

4.2.3 Measurable Concepts Lead To Questions

The previous section defined the measurable concepts that this model uses. This section shows measurable concepts leading to questions. As already stated in section 4.1.1, the questions are divided into two areas. These two areas are further illustrated in Table 7.

Information Axis	Measurable concept	Question
Dynamic Depth	**Dynamic depth for application**	● Maturity question: – How profound are dynamic depths executed inside a DevSecOps CI pipeline? For example, where are dynamic security tests being done? ● Supporting Question: Are there any dynamic tests being done?
Static Depth	**Static depth for application**	● • Maturity question: How profound is static analysis executed inside a Security DevOps CI pipeline? For example, where are static security tests being done?

		• Supporting Question: Are there any static tests being done?
Intensity	**Intensity**	• Maturity question: How intense are tests being executed inside a DevSecOps CI pipeline? For example, what is being tested?
Consolidation	**Consolidation**	• Maturity question: How comprehensive is the process of managing discoveries within a DevSecOps CI pipeline? How are the findings utilised? • Supporting Questions: – Does security scanning detect security problems like vulnerability? – Is there any usefulness of the security problems detected?

Table 7: Measurable Concepts Lead To Questions

Dynamic depth for application

Maturity question: How profound are dynamic depths executed inside a DevSecOps CI pipeline? This question might seem like an obvious question, but it is essential to know where dynamic tests are being applied in the system or application. If a particular area of an application that requires dynamic analysis and checks is omitted, that could be a point of weakness. On the other hand, if not managed well could be exploited by an attacker.
Supporting Question: Are there any dynamic tests being done? This question is fundamental as no dynamic tests or checks will risk-specific vulnerabilities being omitted before a production release.

Static depth for application

Maturity question: How profound is static analysis executed inside a Security DevOps CI pipeline? Again, this question might seem to be noticeable. However, it is relevant to understand how much one has matured within security within the codebase (i.e. from the early stages). These static security checks placed within a DevOps lifecycle are crucial because omitted areas lead to weak points or vulnerabilities in production.

Intensity

Maturity question: How intense are tests being executed inside a DevSecOps CI pipeline? for example, what is being tested? Intensity is one of the most crucial aspects when security tools or checks are added to a CI environment. Here we want to know what is being checked for and how frequently are we doing these checks. For example, we might have a dynamic analysis (i.e. OWASP ZAP), which spiders some part of a web application area. However, it only runs a quick and passive scan once a week, which is limited in capabilities and time, with the goal of aiding agility. One problem is that in a highly sensitive area of an application, a heavy scan might be needed rather than a quick or lightweight scan to ensure that no known vulnerability passes to production. This example shows that an application can have a high level of Depth

(Static and dynamic) yet have a low level of intensity which can still lead to vulnerabilities omissions and Vis Versa. Thus, making intensity an aspect to measure.

Consolidation

Maturity question: How comprehensive is the process of managing discoveries within a DevSecOps CI pipeline? Consolidation aims to understand the results and their usefulness. However, if results are not correctly understood or displayed appropriately, the security check does not add value to the application. For example, if we run an OWASP Zap scan on default settings against a sensitive area of a web app, by default, it will only show high threat vulnerabilities. In this case, if it is just a result of one aspect (high threat vulnerabilities) and does not show other results (like medium and low vulnerabilities). Imagine a situation in which it is added to a CI build process like CircleCI. If one does not indicate a custom logic (e.g. Threshold, Type of vulnerability, Severity of vulnerability) to break the build on, the build will always pass. Thus, making results from this security tool or check useless or pointless. For this reason, it is essential to measure How the results are used.
Supporting Question: Does security scanning detect security problems like vulnerability? As one might already know, if a security tool or check does not do what it is supposed to do, it is rendered useless. So instead, a security tool should be able to determine weak points in a system and give results.

4.2.4 Information Axis Leads to Measurable concepts.

This section covers actual measurements of the improvement in security in the CI when security tools are added. The primary metrics for measurements will be seen in the maturity levels. A maturity method of measurements is used to let a company know where they are and should be. IT security maturity measures a company's adherence to IT security best practice methods and processes [38]. In our case, it can help us measure improvements over some time period. MFAM has five maturity indication levels. The maturity indication levels progress from one to five, where five are the highest and one is the lowest, as shown in Figure 10. In the following sections, we will explain each of the maturity level indicators for a measurable concept.

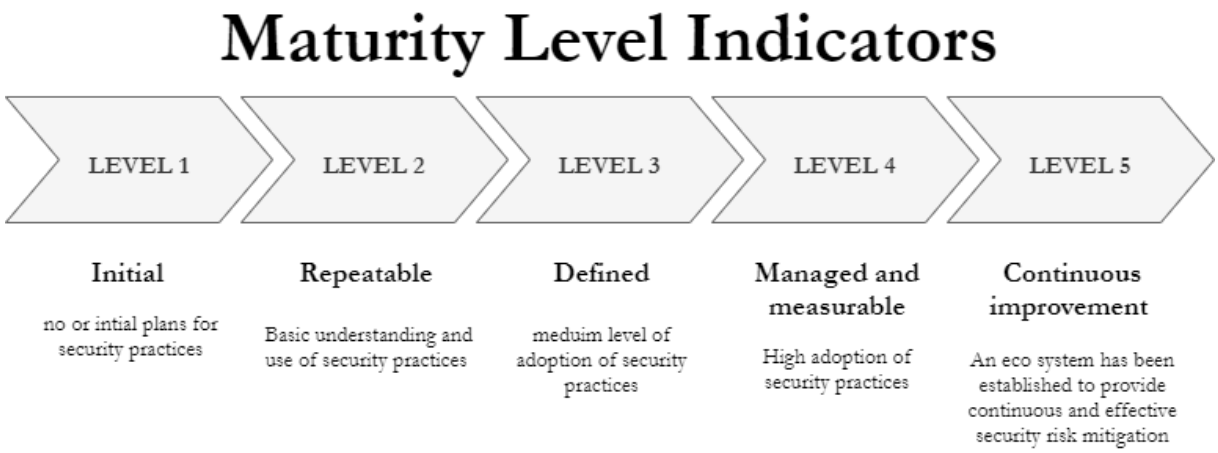

Figure 10: Maturity Level Indicators

Dynamic Depth Maturity Level Indicator

The dynamic depth maturity has five maturity levels ranging from levels 1 to 5. Each level has indicators that have to be reached to be able to achieve a higher maturity level. For example, dynamic tests must be done in the CI pipeline to achieve level 1. Only then can maturity level 2 be checked. These maturity level indicators for each maturity level are given in table 8. ISO27001 2017 and OWASP SAMM standards are used as the reference point for each indicator

Maturity Levels	Maturity Indication
Level 1	● No plans or initial plans are set out for dynamic security checks. ● Absence of procedures or policy which might bring about the dynamic checks
Level 2	● A quick or simple scan is done for a baseline security ○ Reference: ISO27001 2017: 14.2.8, 14.2.3. and OWASP SAMM version 2: V-ST-1-A ○ A scan of the pre-authenticated parts, i.e. public attack surface. ○ No set out requirements for authentication of the scanner with a target. ○ The scan is added to the build or deploys process if the scan time is less than 10 minutes. ○ When more than 10 minutes, the scan is added as a post step in nightly or weekly builds.
Level 3	● The client-side or UI area is covered with a dynamic scan

	○ Reference: ISO27001 2017: 14.2.8, 14.2.3. and OWASP SAMM version 2: V-ST-2-A ○ Spidering the client-side to check dynamic contents like javascript. ○ Dependencies: When there is the usage of different roles ● Integration of all roles in authenticated parts used in an application when scanning, i.e. post-authentication scan. ○ Reference: ISO27001 2017: 14.2.8, 14.2.3. and OWASP SAMM version 2: V-ST-2-A ○ Dependencies: All of Level 2 is covered, i.e. simple scan features ○ All Roles and users authentication groups are scanned ○ The ability for scans to maintain sessions properly. ○ Ability to log out and re-login during tests ○ Different users / roles ○ Spider & scan post-auth
Level 4	● Buried or Concealed endpoints are being found and added to dynamic scan. i.e. coverage of hidden endpoints ○ Reference: ISO27001 2017: not plainly covered by ISO 27001 - too precise and OWASP SAMM version 2: V-ST-2-A ● Custom logics or special parameters are defined while the scanner is run with a fuzzer. i.e. coverage of more input vectors ○ Reference: ISO27001 2017: not plainly covered by ISO 27001 - too precise and OWASP SAMM version 2: V-ST-2-A ● Workflow sequences are defined and checked to be scanned in a well-defined order. i.e. Coverage of sequential operations ○ Reference: ISO27001 2017: 14.2.8, 14.2.3 and OWASP SAMM version 2: V-ST-2-A ○ Not just a simple spidering but checking workflows like (login>> changing account and other business logic) ● o Multiple scanners are used ○ Reference: ISO27001 2017: 14.2.5, 12.6.1 and OWASP SAMM version 2: V-ST-2-A ○ The use of several scanners and spiders help enhance the coverage of vulnerabilities due to the different opportunities by each scanner. ● o Application layers or backends are separately scanned. ○ Reference: ISO27001 2017: not plainly covered by ISO 27001 - too precise and OWASP SAMM version 2: V-ST-2-A ○ Webservices like SOAP which are internal, are scanned. i.e. directly scan backends

	○ Parameter positions like JSON and XML should be detected and scanned. (IAST might help) ○ Check proxies that are part of the backend service calls
Level 5	● No missing parts with coverage tool in the application, i.e. Coverage analysis ○ Reference: ISO27001 2017: not plainly covered by ISO 27001 - too precise and OWASP SAMM version 2: V-ST-2-A ● Service to service communications are checked and dumped. i.e. Service to service communications ○ Reference: ISO27001 2017: 14.2.8, 14.2.3 and OWASP SAMM version 2: V-ST-2-A

Table 8: Dynamic Depth Maturity Level Indicator

Static Depth Maturity Level Indicator

The static depth maturity level follows the same pattern as the dynamic depth. It also ranges from levels 1 to 5, and a level can only be achieved when the previous levels are reached. These maturity level indicators for each maturity level are given in table 9. The goal of an organisation here is to reach a maturity level of five. ISO27001 2017 and OWASP SAMM standards are used as the reference point for each indicator

Maturity Levels	Maturity Indication
Level 1	● No static code analysis ● If the code works, do not fix it. ● Plans are being made to add these tools to use.
Level 2	● Check for dependencies with known vulnerabilities: ○ Reference: ISO27001 2017: 12.6.1 and OWASP SAMM version 2: V-ST-2-A ○ Test application server-side dependencies. E.g. JAR files ○ Test application Client-side dependencies, e.g. NodeJS frameworks.
Level 3	● Important aspects of code are scanned ○ Reference: ISO27001 2017: 12.6.1 and OWASP SAMM version 2: V-ST-2-A ○ A SAST tool to scan important aspects of the applications codebases, e.g. String matching algorithms, code changes code in a sprint, developed custom frameworks. Possibly large multi-project maven projects - at least backend and frontend.

Level 4	● Scan the complete applications source code ○ Reference: ISO27001 2017: 12.6.1 and OWASP SAMM version 2: V-ST-2-A ○ Just like level 3, Scan all components code for vulnerability patterns, E.g. All projects of a JAVA project.
Level 5	● Automatic removal of code duplicates ● o Analysis of compliance to code style guides, i.e. Stylistic ○ Reference: ISO27001 2017: 12.6.1, 14.2.1, 14.2.5 and OWASP SAMM version 2: V-ST-2-A ○ Ensure code rules are met like indent rules. ● Static analysis of all libraries and components as well as critical third-party dependencies used in the application. Open-source components should be part of the regular scanning events. When the component is open-source, it should be included in regular scanning activities. For closed source components, it is considered when using code scanners that operate on binaries.

Table 9: Static Depth Maturity Level Indicator

Intensity Maturity Level Indicator

The intensity maturity level follows the same pattern as the static and dynamic depth. Thus, the maturity levels are from levels 1 to 5, in which level 5 is the highest. A maturity level needs to be reached for another to be checked. To develop these maturity levels, ISO27001 and OWASP SAMM are used as the reference point. The maturity level is shown in table 10

Maturity Indication Levels	Description
Level 1	● No scanners in place or only plans for scanners have been put in motion.
Level 2	● Default settings for intensity: The intensity settings are not set to save time ○ Reference: ISO27001 2017: 12.6.1, 14.2.1, 14.2.5 and OWASP SAMM version 2: V-ST-2-A ○ Dynamic Checks: Only passive scanning. ○ Static Checks: Using default rules of the scanners and integrating them into your CI pipeline ● High test intensity: A deep scan for both static and dynamic with a low confidence threshold and a high test intensity is done.

	○ Reference: ISO27001 2017: 12.6.1, 14.2.1, 14.2.5 and OWASP SAMM version 2: V-ST-2-A
Level 3	● Tests that are not needed are diactivated. (lightweight scan) e.g. no SQL database, so SQL Injection tests are removed. ○ Reference: ISO27001 2017: 14.2.3, 14.2.8, 14.2.9 and OWASP SAMM version 2: V-ST-2-A ● Regular test: On each interval or push, security tests or scans are being done ○ Reference: ISO27001 2017: 12.6.1, 14.2.1, 14.2.5 and OWASP SAMM version 2: V-ST-2-A
Level 4	● Use heavyweight scanning and high-intensity scanning options on essential parts of the application: ○ Reference: ISO27001 2017: 12.6.1, 14.2.1, 14.2.5, 14.2.2, 14.2.3 and OWASP SAMM version 2: V-ST-2-A ○ Dynamic checks: risker custom test profiles and concepts are made. i.e. depending on the time per scan and intensity, the test can be done on every commit, every night, week or month. ○ Static checks: Find security bugs and set "threshold" to "low" to "Blocker."
Level 5	● A customised rule set shout be used for dynamic scans.

Table 10: Intensity Maturity Level Indicator

Consolidation Maturity Level Indicator

The consolidation maturity level follows the pattern of the static, dynamic and intensity maturity levels. It ranges from level 1 to 5, where level 5 is the highest and level 1 lowest. Thus, a maturity level needs to be achieved for another maturity level to be checked. These indicators are shown in table 11. ISO27001 2017 and OWASP SAMM standards are used as the reference point for each indicator.

Maturity Indication Levels	Description
Level 1	● Results are not used or not used correctly (ignored) ● Lack of understanding of results or no results

Level 2	● Quality gates should be defined. ○ Reference: ISO27001 2017: not plainly covered by ISO 27001 - too precise, 12.6.1, 16.1.4 and OWASP SAMM version 2: I-DM-2-A ○ Setting an alert to high or critical vulnerability at the start to avoid overload at the beginning ● Simple treatment of false positives ○ Reference: ISO27001 2017: not plainly covered by ISO 27001 - too precise, 16.1.6 and OWASP SAMM version 2: I-DM-2-A ○ False positives are suppressed to not show in the following tests. ○ Many tools have the capability of suppressing false positive ○ The use of a vulnerability management system is another way. ● High severity defects are treated. ○ Reference: ISO27001 2017: 12.6.1, 16.1.4 and OWASP SAMM version 2: I-DM-2-B ○ High severity Vulnerabilities and higher are added in the quality gate ○ Both static and dynamic scanners generate readable defect reports, e.g. published in CircleCI
Level 3	● Vulnerability issues are part of the development process ○ Reference: ISO27001 2017: not plainly covered by ISO 27001 - too precise, 16.1.4, 16.1.5, 16.1.6 and OWASP SAMM version 2: I-DM-2-B ○ Vulnerabilities are tracked with an issue tracker like Jira, teams, slack ○ Custom logic to break the build depending on the severity, confidence level and type of vulnerability. ○ Useful remediations are provided to developers ● Defects with medium or middle severity are treated ○ Reference: ISO27001 2017: 12.6.1, 16.1.4 and OWASP SAMM version 2: I-DM-2-B ○ Middle or medium Vulnerabilities and high are added as part of the quality gate. ● A vulnerability management system is used ○ Reference: ISO27001 2017: not 16.1.4, 12.6.1, 16.1.3, 16.1.5, 16.1.6 and OWASP SAMM version 2: I-DM-1-B ○ Track all vulnerabilities from severaL tools in one source or management system

Level 4	● Proper and better handling of defect virtualization ○ Reference: ISO27001 2017: 8.2.1, 16.1.4, 8.2.3, 8.2.2 and OWASP SAMM version 2: I-DM-1-B ○ Defects can be visualised by the corresponding team project that can fix the problem, e.g. OWASP DefectDojo ○ Duplicate discoveries from diverse scanners are de-duplicated and consolidated. (use a vulnerability management system to aid in this)
Level 5	● Defect tickets can be reproduced ○ Reference: ISO27001 2017: 8.2.1, 16.1.4, 8.2.3, 8.2.2 and OWASP SAMM version 2: I-DM-2-B ○ When Vulnerabilities can be reproduced, it gives employees or teams a better understanding of Vulnerabilities and fixes become better. ● o Treatment of all defects ○ Reference: ISO27001 2017: 12.6.1, 16.1.4 and OWASP SAMM version 2: I-DM-2-B ○ Check code coverage of security tests to see that all aspects are covered. i.e. finding white spots which are untested. ○ Analyse and check where the scanner should focus more based on results

Table 11: Consolidation Maturity Level Indicator

4.3 Conclusion

In this chapter, we developed the MFAM model for the measurements of security improvement in the CI, which involves five main aspects. The model was created to help answer RQ1.1 on the measurement of security improvement when tools or checks are added. In the following chapter, we will be applying this model in practice with a case study on BiZZdesign.

5 MFAM in practice: A Case Study at BiZZdesign

In the last two chapters, we looked at the security tools and checks that can be added to a CI pipeline. In addition, we also developed a model to elaborate on how security improvements can be measured in a CI pipeline when security tools are added. This chapter builds a case study based on the previous two chapters to see how the measurements can be made in BiZZdesign.

BiZZdesign is Dutch enterprise architecture and BPM software-development tools vendor and consultancy firm founded in 2000 by Henry Franken [39]. At BiZZdesign, many DevOps tools are used to manage their software and DevOps teams. As shown in Figure 7 on BiZZdesign's Pipeline Architecture, many tools are used from GitHub to AWS Cloud environment. The company uses these tools to build and increment their main software applications product deliveries like Enterprise Studio and HoriZZon. In this case study, we focus on the CI pipeline within BiZZdesign. The research question for this case study is RQ1.2 being *"In practice, is it possible to measure security improvements when security tools are added to a CI pipeline?"*

This case study will be done in two phases. Phase one will evaluate and measure the current architecture with our MFAM model, i.e. the current state of affairs at the BiZZdesign CI pipeline. In the second phase, we will add some security tools to the CI and finally evaluate them.

Note: Due to a Non- disclosure agreement, this case study does not go into critical and sensitive information that might harm the organization. This disclaimer means that this chapter will focus on the measurement model in practice and will not go in-depth on the organization's architecture but instead focus on a broader view. However, Figure 7 gives a broad overview of what the architecture looks and feels like for better understanding.

5.1 Phase One: Evaluation Of The CI Pipeline At BiZZdesign With MFAM

To measure improvement, one must understand what was there or where one is and after changes are made, we check to see what improved. From the architecture of BiZZdesign in Figure 7, we can see that the developers push codes to Github. This code then triggers CircleCI on any changes made, running several build jobs and test jobs. When all build and test jobs are passed, the build artefact is sent to the Amazon ECS. Finally, if the build or test job fails, it sends the test result back to the release branch. This workflow summarises how the CI in BiZZdesign works, which is illustrated in Figure 11.

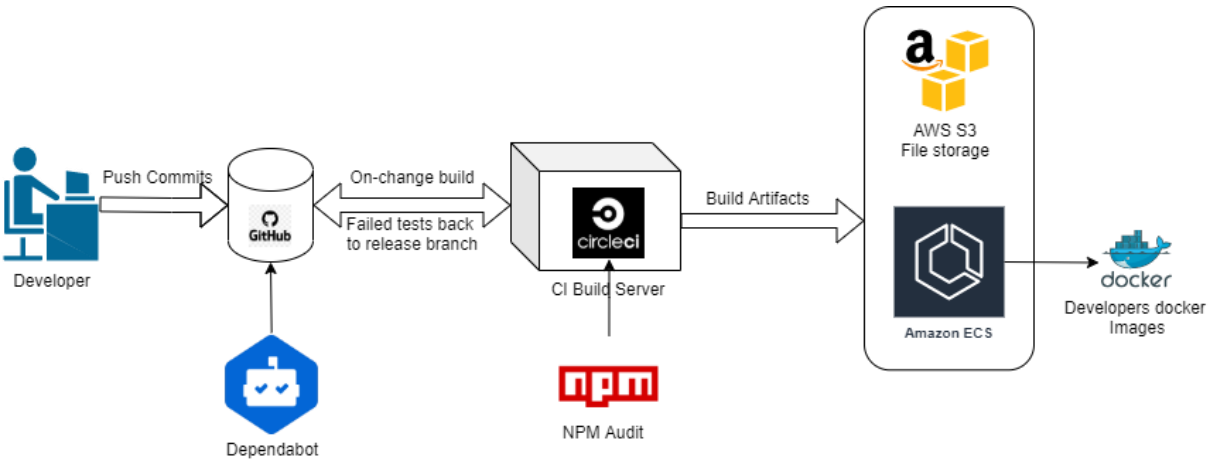

Figure 11: Phase One - Some initial security tools in Bizzdesign CI Pipeline

From Figure 11, we can see an example of two security tools in the pipeline of BiZZdesign, namely npm audit and Github DependaBot. Next, we will further explain what these tools do.

- **Github DependaBot:** This tool aims to help keep dependencies up to date automatically. Many organizations use open sources dependencies to build their software and applications. As time goes on, some of these dependencies require updates due to known vulnerabilities. These known vulnerabilities might have been discovered in a previous version or release. As codebases can run into the millions, updating these dependencies can be a pain for developers. DependaBot alleviates that pain by updating one's dependencies automatically, so one can spend less time updating dependencies and more time building [40]. DependaBot is a tool that comes under Static Code Analyzer (SCA). It does not scan the code for vulnerabilities, but it scans source files and binaries for digital signatures, i.e. showing vulnerable dependencies that need updates.
- **NPM audit:** NPM stands for Node Package Manager, a package manager for Node.js and allows Javascript developers to share node modules [41]. This tool is added as a testing tool in CircleCi within BiZZdesign. The audit is a command within NPM that runs a security audit on accessed package dependencies for security vulnerabilities. This command will submit a description of the various dependencies configured in the project to the default registry of the project and then requests for the report. If the requested report contains any vulnerabilities, then the impact and remediation for the vulnerability will be calculated accordingly. In order to apply the remediations to the package tree, the argument fix must be provided [42].

Now that the current security tools added within the CI pipeline at BiZZdesign have been seen, the following steps will apply the MFAM model. Applying the model now will help us understand where we are and where we can be.

5.1.1 Applying MFAM Metrics To Phase One

To apply the MFAM model, we must first set up a timeframe for this analysis. The timeframe will help to act as a guide to help the reader understand the progress made and the following plans. This timeframe plan is seen in Figure 12. This plan is only there as a guide and is, however, not strictly followed in practice.

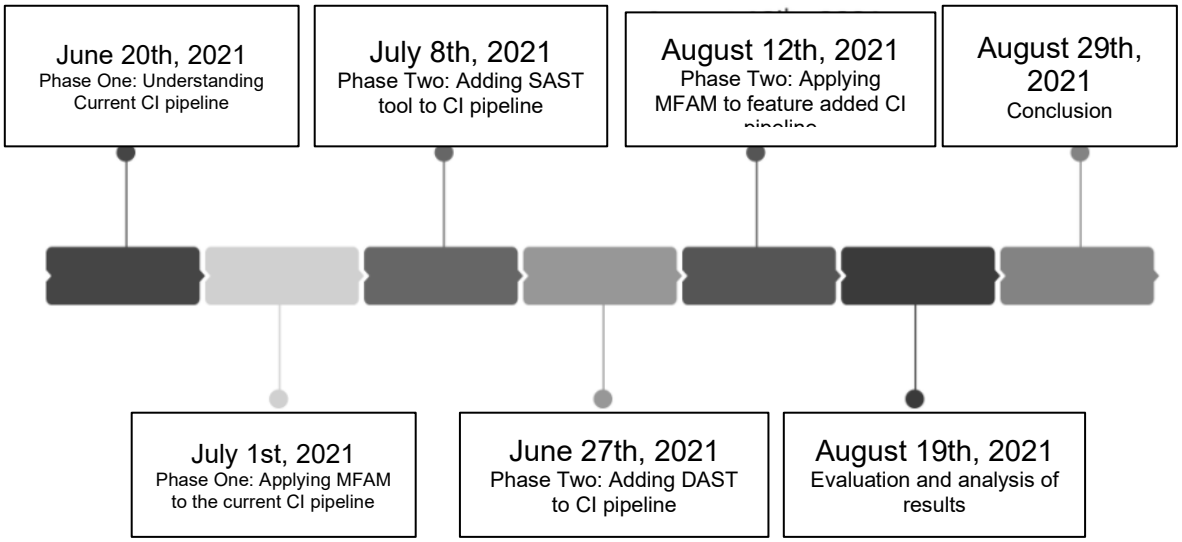

Figure 12: Case Study Timeline

It is crucial to understand that no additional security tools are integrated into the CI pipeline in this phase. The goal here is to measure what is already in the CI pipeline. We will take each measurable concept and see what results we can draw from it. To ensure that all areas of the MFAM model are covered, we must ask questions. These questions are rather informational questions or knowledge bases that help us properly understand the current situation and give an accurate maturity profiling, i.e. Supporting Questions. The tool's coverage and configuration analysis are done with follow-up interviews with the product manager and lead software engineer to help answer the questions. *It is essential to know that the answers to the questions will be broad due to a non-disclosure agreement.*

5.1.2 Dynamic Depth For Application

Maturity Level 1:

Are there any dynamic checks being applied in the CI pipeline? The answer is that no Dynamic checks are being done as part of the current build. However, checks are being done through penetration testing within a specific timeframe. This answer was confirmed by the product manager and lead developer within the interview. This information means that the maturity indication is one (1) within the dynamic depth for applications. In this case, there is no need to go further with the questions.

Result: The maturity level given in this measurable concept is set to level one, as can be seen in table 12.

5.1.3 Static Depth For Application

Maturity Level 1: This level aims to know if any plans or any dynamic checks are being done.

Are there any static checks being applied in the CI pipeline? The answer to this question is Yes; within the code, test and build phase. In the above section, we have already seen that npm audit and Github DependaBot is being used at the codebase (Github) and test/build (CircleCI) phase. Therefore, this statement helps confirm the answer to this question. Further confirmation was then done by consulting the product manager and lead developer. This answer means we can move on to check level two.

Maturity Level 2:

Where are the static checks being applied? i.e. are they being applied on the client or server-side components (e.g. backend/middleware)? The answer to this question is yes, they are being applied at the server-side and client-side components. To confirm this, we took a look at where npm audit and Github Dependabot within the code, test and build phase. The two have the function of ensuring that dependencies with known vulnerabilities are avoided. They do this through automatic upgrades and notifications to the developers. Finally, further confirmation is gotten from the responsible individuals that this is indeed correct and valid. Thus, Npm audit works on the client-side, while dependable works on the server-side. With this answer, we can then move on to the next maturity level.

Maturity Level 3:

Are important parts of the application codebase being scanned for vulnerable patterns? The answer to this question is No. To confirm this answer, we took a look into the possible tools that could perform this kind of action with the CI pipeline. Unfortunately, we could not find a reliable security-related static analysis tool. From the lead software developer, specific static analysis tools were in place. However, non was a dedicated security tool to catch vulnerabilities directly. This response means that the maturity level of three has not yet been fully reached, and no further questions are required for other levels.

Result: The maturity level given in this measurable concept is set to level two, as can be seen in table 12.

5.1.4 Intensity

Maturity Level 1:

Are there any security checks or scanners in place in the CI pipeline? The answer to this question is Yes; within the code, test and build phase. This answer is confirmed from the above section on static depth. Without a security scanner or a check, there cannot be any intensity. The answer means the next level can now be achieved.

Maturity Level 2:

Are the tests being set in a way to save time? The answer is yes. The product manager and developers verified this answer. Also, looking at the configuration of the scan being done, the implementation of caching is used. Caching is a method of storing data so that they become

readily accessible on future requests. Therefore, we can conclude that configurations are done to save time for developers.
Is the intensity of the scan high and the confidence threshold low? The answer to this question is Yes. Looking at the configurations of NPM audit and DependaBot, the confidence level is at default set to low. A confidence level is a numeric value given to a label when evaluating an issue. The confidence level is not explicitly set in npm audit or DependaBot but can also be determined by what is scanned. The advantage of this is that all possible security vulnerabilities for either tool will be shown. However, this also means that there could be a lot of false positives.

Furthermore, Dependency security scans usually have no false positives as it scans dependencies for those with reported known vulnerabilities. On the other hand, the scan is also intense as it checks for all dependencies, which are known to have vulnerabilities. This answer means that the maturity next level can be achieved

Maturity Level 3:

Are the tests being done regularly? The answer is yes. The frequency of the tests is done on every code pushed.
Is the test being accustomed to only test what is being used? (e.g. no SQL injection scan because firebase is being used) Unfortunately, the answer is no. The configurations made are instead default rather than specific. With this, we can't be moving to the next level.
Result: The maturity level given in this measurable concept is set to level two, as can be seen in table 12.

5.1.5 Consolidation

Maturity Level 1:

Are the tests giving any results? The answer is yes. Each tool used provides results regarding the security problems and issues in a way that is perceived to be understandable to its users.

Maturity Level 2:

Is a simple quality gate defined? The answer is yes. This answer was established after a corresponding review of each tool's configuration. The treatment of defects is set to high or critical, meaning only high vulnerabilities will be blocked at the time of release.
Is there a readable and straightforward visualization of defects? Yes, there is. This visualization is easily seen within GitHub and CircleCI. For example, the DependaBot and NPM audit showed a straightforward visualization of vulnerabilities, i.e. its type and severity.
Is there a simple treatment for false positives? The answer is no. There is no current set definition to suppress false positives. The only thing that could be done is to ignore an alert rather than marking it as a false positive. Due to this, there is no need to move to the next stage.

Result: The maturity level given in this measurable concept is set to level two, as can be seen in table 12.

A summary of these results is seen in table 12. The summary shows what has changed since the addition of the new security tools. The question one might have at this point is, “Was

any improvement shown?". The next step covers evaluating the results of these MFAM metrics and the effects of the security additions on agility.

Phase	Information Axis	Measurable concept	Required maturity level based on inherent risk estimation	Actual maturity level indication
Phase One	**Static and Dynamic Depth**	**Dynamic depth for application**	3	1
		Static depth for application	3	2
	Intensity	**Intensity**	3	2
	Consolidation	**Consolidation**	3	2

Table 12: Phase One - Maturity profile of BiZZdesign with tested security tools

5.2 Phase Two: Adding Security Checks To CI Pipeline At BiZZdesign

This phase is crucial to how our research questions can be answered. Chapter 4 looks at the security options or tools such as SAST and DAST that could be added to the CI pipeline. We saw that many tools could be used in each option type. After some consultation with the respective parties at BiZZdesign, a decision was made to add SonarCloud for SAST analysis and OWASP Zap for DAST to the CI pipeline for test purposes. In the following steps, SonarCloud and OWASP Zap are explained, as well as their setup. Finally, an application of the MFAM metrics is made. The illustration of these additions can be seen in Figure 13. The Difference between figures 13 and 11 is the addition of SonarCloud and OWASP ZAP to the already existing tools (NPM Audit and GitHub DependeBot).

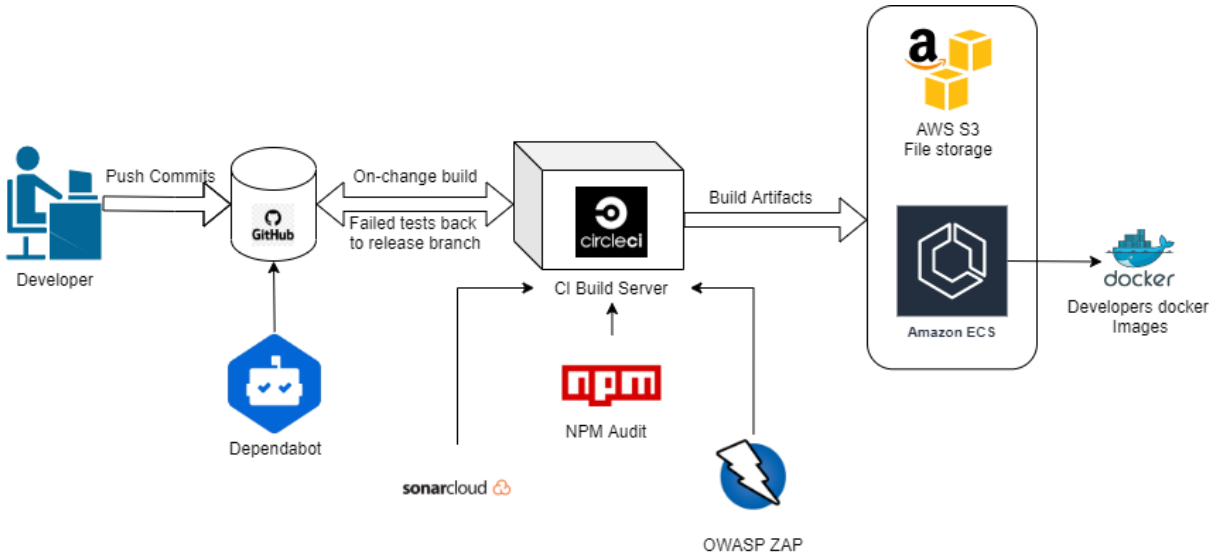

Figure 13: Phase Two - Addition of SAST and DAST to Bizzdesign CI Pipeline

5.2.1 What is SonarCloud?

SonarCloud is the leading online service to catch Bugs and Security Vulnerabilities in your Pull Requests and throughout your code repositories [44]. It is freely accessible for open source projects, but a paid plan is needed for private projects. It has a wide range of languages it supports like C#, C++, Python. As the name implies, it is cloud-based, which means analyses are done SonarClouds cloud hosting.

5.2.2 How does SonarCloud Work?

SonarCloud has several ways in which it works. One way is manually adding code to SonarCloud and running it only when one sees fit. The other way which is mainly used is pairing SonarCloud with an existing Cloud-based CI workflow. This pairing allows features like branch analysis, pull request analysis, and all build analysis on code. Usually, these pairings are triggered based on a YAML configuration file used in a CI system, e.g. CircleCI, TravisCI and more. This trigger is a plugin named SonarScanner. SonarScanner reads the codebase to understand the parameters, code paths and code type (e.g. python, Java). It then sends the code to the SonarCloud server for analysis. An illustration of what happens when SonarCloud is triggered can be seen in Figure 14.

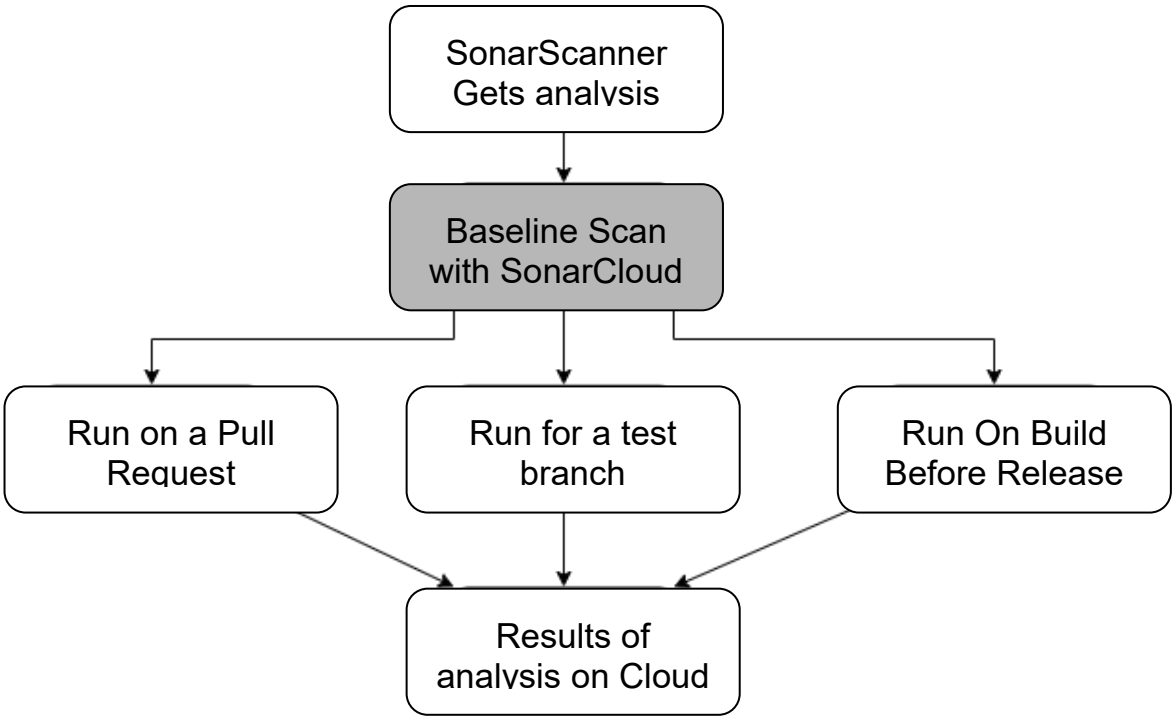

Figure 14: The workflow of Sonar Cloud In a CI pipeline

5.2.3 How was SonarCloud Run?

Running SonarCloud on CircleCI within BiZZdesign, the following command was used in the YAML file for CircleCI. The following configuration is for a C++ project run. Upon execution, SonarCloud determines if it is an analysis for a branch or build or pull request. This helps to analyze specific areas of the application. ***Note that the values of the parameters are not provided for security and privacy reasons.***

```
sonar-scanner \ #Trigger command
-Dsonar.projectKey={Company.projectkey}. \ #the project key set for
analysis
-Dsonar.sources={directory.to.source} \ #the directory to scan or
analysis
-Dsonar.host.url={sonar.server.url} \ #SonarCloud Server URL
-Dsonar.login={my.login.key} \ #the api login token key
-Dsonar.projectVersion=1.0
-Dsonar.cfamily.build-wrapper-output= {file/path/to/build-
wrapper/json/file}
-Dsonar.language=c++
-Dsonar.cfamily.threads={no.of.process.threads}
-Dsonar.sourceEncoding=UTF-8
-Dsonar.exclusions={some.file.type. excluded}# example **/*.java
```

5.2.4 What is OWASP ZAP?

Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project (OWASP) [45]. It is a robust open-source security tool that an international team volunteer actively maintains. It helps individuals and organizations in their web applications by finding security vulnerabilities while testing and developing their applications.

5.2.5 How does OWASP ZAP Work?

ZAP works as a man-in-the-middle proxy. It stands between the tester's browser and the web application to intercept and inspect messages sent between the browser and web application, modify the contents if needed, and forward those packets to the destination [45]. It can work as a daemon or a standalone application with the provision to work on several Operating systems and Docker (so you are tied to a single OS). An illustration can be seen in Figure 155.

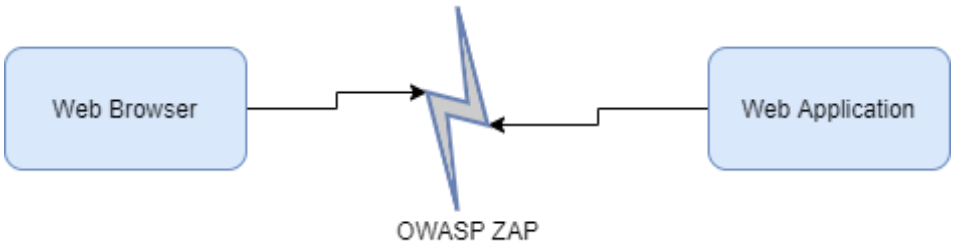

Figure 15: OWASP Zap working as a man in the middle proxy.

5.2.6 How is OWASP ZAP run?

Running OWASP Zap is usually done as part of the smoke test jobs. For example, at BiZZdesign, it was done in CircleCI at the final testing stages. As already known, CircleCI runs with configuration information for a set YAML file. This configuration file is where OWASP Zap configuration details are used. One easy way is to run it over Docker as follows:

```
docker pull owasp/zap2docker-stable
            (
              docker run  --name zap -i owasp/zap2docker-stable \
                zap-cli quick-scan \
                --self-contained \
                --start-options '-config api.disablekey=true' \
                --spider\
                "https://target-url
                -l Medium

              if [$? -ne 1]; then exit 0; else exit 1; fi;
            )
```

5.2.7 Applying MFAM metrics to Phase Two

SonarQube and OWASP Zap have been integrated into a test environment as our SAST and DAST tools. The next step is to measure what has improved with our MFAM metrics. Then, like in the first phase, knowledge base questions are asked, and answers are given based on the results from these security tool additions. The questions asked are not the main maturity questions but instead supporting questions that help one get the maturity level required for a measurable concept. Following this section is an Evaluation of results in which results of both phases are analysed.

5.2.8 Dynamic Depth For Application

Maturity Level 1:

Are there any dynamic checks being applied in the CI pipeline? The answer is yes. From the previous section, it is seen that OWASP Zap, which is a DAST tool, is added to the CircleCI pipeline at BiZZdesign. This answer means that the maturity Level has now been reached, and we can move over to the next level..

Maturity Level 2:

Is there a simple dynamic scan being applied on a targeted test or pre-release environment? i.e. Where are the dynamic checks being applied? The answer is yes. Within the testing environment provided, a spider and scan of the public attack surface were approved to be done. This check is done with ZAP quick scan, e.g. `zap-cli quick-scan targetURL`. ZAP scans the pre-authenticated parts of the targeted test environment within BiZZdesign. It is added as part of the build process in CircleCI as its tests fall under the 10min mark. This means it ticks all the boxes at this level

Maturity level 3:

Are the client-side or UI areas covered with dynamic scans? The client-side is usually areas that contain client-server components like javascript. Within the spider, it is evident that server-side components are being scanned when looking into the spider coverage. For example, while using the ZAP scan, the command `zap-cli quick-scan -- ajax-spider targetURL` is used to spider for areas with client-side components like the login page.

Are all post-authenticated parts with user roles covered in the scan? The scan was only tested on a few user rows authorized to use in the test environment—for example, a developer or tester. This implies that the answer to this metric is not passed. Thus, the next level cannot be reached.

5.2.9 Static Depth For For Applications

In this section, it is already known that maturity level two has been achieved with the initial security options put in place. The reason for this is that there was no dedicated static code analysis tool for security to scan essential aspects of the codebase. However, an addition of a static code analysis tool (SonarCloud) has been done. Therefore, to avoid redundancy, we go straight to maturity level three, which was not achieved in phase one, to find out what might have changed.

Maturity Level 3:

Are important parts of the application codebase being scanned for vulnerable patterns? SonarCloud has been integrated into the CI test environment. In the section on how SonarCloud was running, there is an add-on or plugin called SonarScanner. SonarScanner has an input command which helps one scan only certain areas they see fit to be necessary. As the configuration used cannot be added to this thesis report, an example of how this configuration might be is as follows:

- ```
  sonar-scanner \ #Trigger command
  -Dsonar.projectKey={Company.projectkey}. \ #the project
  key set for analysis
  -Dsonar.sources={directory.to.source} \ #the important
  codebase directory to scan or analysis
  ...
  Other essential configuration specs needed, Like login
  token
  ...
  -Dsonar.exclusions={some.file.type. excluded}# example
  **/*.java
  ```

Looking at the configuration command above, it becomes evident that the commands `Dsonar.sources` and `Dsonar.exclusions` let SonarCloud know the crucial parts of the code it should look at. `Dsonar.exclusions` is a command which tells SonarCloud the type of file it should not use. It is used in a situation where certain file types are not crucial to the main workflow of the system or application. On the other hand, `Dsonar.sources` lets SonarCloud know the file directories it should scan, and any directory not included under that parent directory is ignored.

This response means that the maturity level of three has now been reached, enabling the next maturity level to be checked.

Maturity Level 4:

Are all parts of the application codebase being scanned for vulnerable patterns? The answer is no. The codebase is enormous, and many projects are being run. The SonarCloud is
  ```

still in the testing stages, and it is not being run on the whole codebase but rather on vital areas. Due to the factors mentioned above, the maturity level has not been reached. Since the maturity level has not been reached, it is. Essential to move to the next step rather than to the next maturity level.

5.2.10 Intensity

In Phase One, the maturity level of two was reached because all questions in maturity level three for intensity were not reached. However, the DAST and SAST security tools have also brought additions to the answers to the questions asked in maturity level two. In order to avoid redundancy, the maturity levels with no significant change are not covered I.e. maturity one has been reached as scans and tests are being done.

Maturity Level 2:

Are the tests being set in a way to save time? As an addition to caching in phase one, SonarCloud and OWASP Zap are set to save time. In SonarCloud, the multi-threading command is used and the exclusion command to save the time of scan execution. For example, we run the following commands with the SonarScanner plugin when running a CPP project to help save time.

- `Dsonar.cfamily.threads={no.of.process.threads}`
- `Dsonar.exclusions={some.file.type. excluded}# example **/*.java`

In OWASP Zap, it is rather different. Time is saved based on the type of scan that is being done. In our case, a quick, simple scan is done. For example, zap-cli quick-scan runs a scan in less than five minutes, i.e. depending on the added feature commands like `-ajax-spider` or `-active-scan`.

Is the intensity of the scan high and the confidence threshold low? SonarCloud and OWASP zap confidence level is by default low. Just like it is stated in phase one, a low confidence threshold brings about false positives. In a situation that an alert in SonarCloud and OWASP Zap is indeed a false positive. It can be fixed in both tools by setting the confidence level to false positive. This acts as a possible fix to this issue. However, it is only possible to do this on the GUI aspect of the OWASP zap.

Maturity Level 3:

Are the tests being done regularly? The frequency of the tests has not changed. It is still being done on every code pushed. However, this comes at the cost of agility (more on this later)

Is the test being accustomed to only test what is being used? (e.g. no SQL injection scan because firebase is being used) We had already seen in the setup of SonarCloud that the Dsonar.exclusions and the Dsonar.sources command help us custom the scan to test for only certain types of files. There is also a rule set within SonarCloud that can help in a more customized setup. However, it was not used during the testing. In OWASP Zap, it was instead not used during testing as this rule set might take time to develop, which might heavily affect the timing of this thesis. The maturity level of three has now been reached, making it possible to move to the next level.

Maturity Level 4:

Are the tests heavyweight? Unfortunately, the answer is No. SonarCloud did have a heavyweight scan as the severity threshold is set to critical and code coverage quite extensive. However, in OWASP Zap, the scan on the test target within BiZZdesign was a lightweight, quick scan. Which means this maturity level has not yet been met. The next step is now moved to rather than the next level.

5.2.11 Consolidation

Maturity Level 2:

Is a simple quality gate defined? In SonarCloud, there is what is referred to as the quality gateway. When a scan is done, the quality gateway must be passed to ensure a set standard. An example of one of the quality gates used in SonarCloud can be seen in Figure 16. From the figure 16, it is immediately seen that specific criteria are used as defined factors to be passed to achieve the quality needed. In OWASP Zap, it is somewhat different. The quality gate is usually set with the custom logic on the type of severity to block the build, e.g. only on high vulnerability findings.

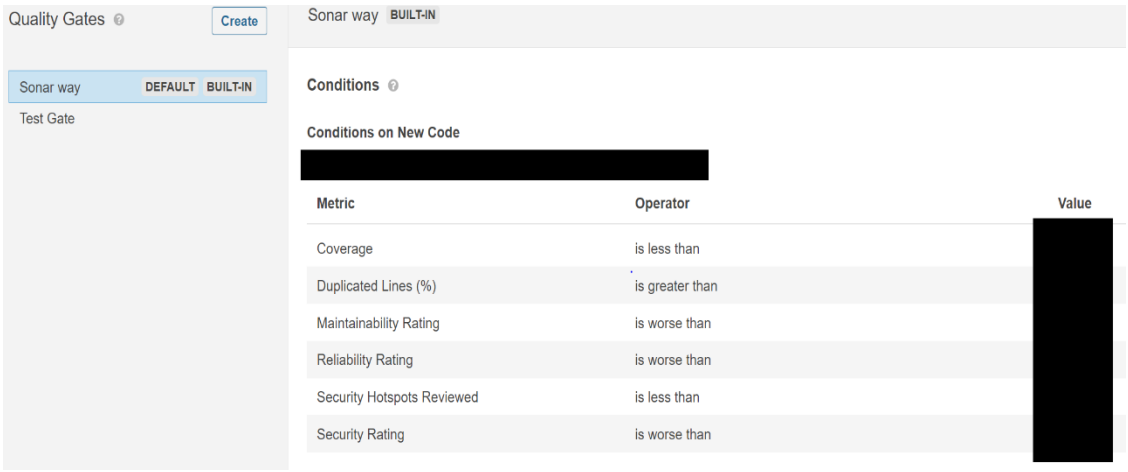

Figure 16: SonarCloud Quality Gate Configuration Dashboard

Is there a readable and straightforward visualization of defects? In addition to phase one, there is indeed a straightforward visualization of defects for newly added tools. An example of one of the visualizations for SonarCloud is shown in Figure 17

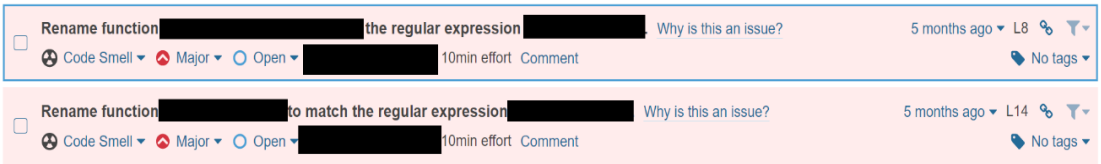

Figure 17: SonarCloud Vulnerability visualization Dashboard

Is there a simple treatment for false positives? With the newly added tools, it then becomes possible to treat false positives. In OWASP Zap, this can only be done from the User interface. It is done by right-clicking on the alert and marking it as a false positive. In SonaCloud, it is done by clicking on the "Open" option button, as shown in Figure 17.

All questions for maturity level 2 have been answered positively, which means the subsequent maturity level compliance can be checked.

Maturity Level 3:

Are the vulnerability issues tracked within the development process? E.g. teams or Jira. Just like it is shown in Figure 7, Jira is part of the development process in BiZZdesign. Jira is an issue tracking system to aid teams and organisations in tracking issues within their development process. This means that when an issue comes up within the development process, it can easily be tracked. It applies to SonarCloud, OWASP Zap, NPM audit and DependaBot. However, there might be limitations to what it tracks. For example, if SonarCloud gateway fails during a build. The build then fails. This is possible to track in Jira as it is connected with the CI pipeline (CircleCI). However, if you want specific issues like a critical vulnerability to show in Jira automatically from SonarCloud, it only then becomes possible by integrating SonarCloud directly with Jira.

Are defects with middle-classed severity or higher being handled? The answer is no. As these security additions are still in the testing stages and have not been fully implemented into the main workflow, only defects with critical vulnerabilities are considered for our test case.

Are there any vulnerability management systems being used to manage defects from different tools? The vulnerability management systems can help with handling false positives, but there is no Vulnerability management system added. Thus the next maturity level cannot be achieved.

A summary of these results is seen in table 13. The summary shows what has changed since the addition of the new security tools. The question at this point is, "Was any improvement shown?". The next step covers evaluating the results of these MFAM metrics and the effects of the security additions on agility.

Phase	Axis	Measurable concept	Required maturity level based on inherent risk estimation	Actual maturity level indication
Phase Two	**Static and Dynamic Depth**	**Dynamic Depth for application**	3	3
		Static Depth for application	3	3
	Intensity	**Intensity**	3	3

	Culture and Consolidation	**Consolidation**	3	3

Table 13: Phase Two - Maturity profile of BiZZdesign with added security tools

5.3 Evaluation Of Results

In the last two sections, the level of security checks brought was measured with MFAM. The goal of this maturity model is not just to answer questions on measurements but to instead discover to what extent the improvement has been made. The Goal question metrics (GGM) is a simple model that can measure security improvements. However, the primary issue is that it does not show the extent of the improvement. It instead shows that there has been an improvement. This is one of the problems which the MFAM model fixes. A graph was drawn up from the tables presented in phases one and two to visualise the results. This graphical representation can be seen in figures 18 and 19.

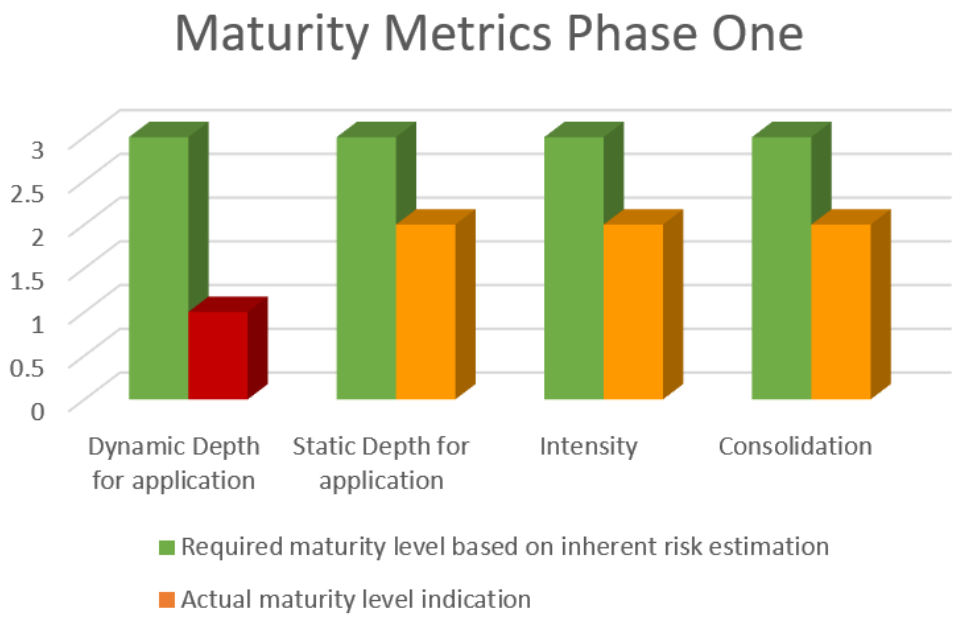

Figure 18: Phase One - Maturity profile of BiZZdesign with before security tools

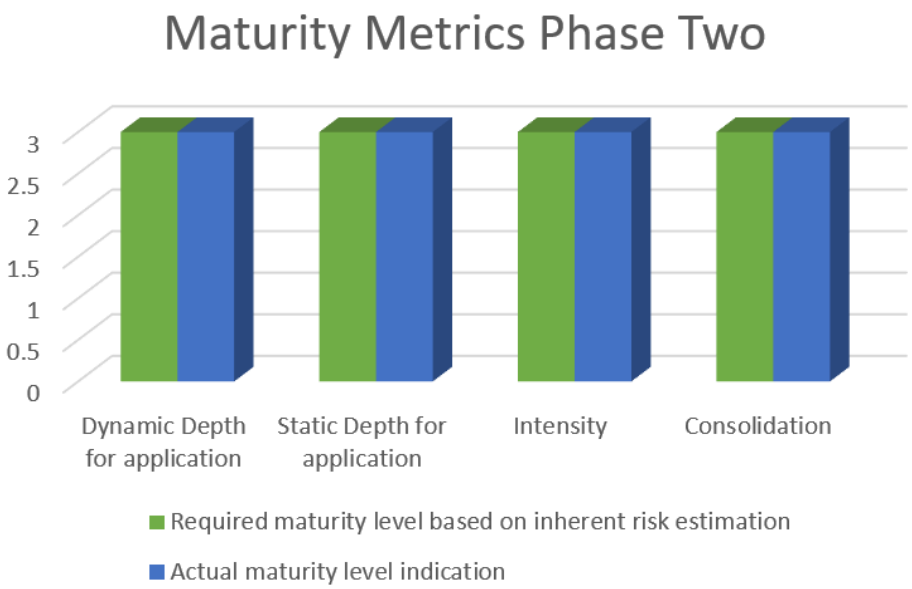

Figure 19: Phase Two - Maturity profile of BiZZdesign with added security tools

5.3.1 MFAM Questions Answered

From the MFAM model architecture in chapter five, questions have been asked under measurable concepts which need to be answered. These questions are answered from the results of the metrics from these two phases. In the next step, the answers to these questions are provided. Finally, each measurable concept is taken at a time and evaluated.

DYNAMIC DEPTH FOR APPLICATIONS QUESTION ANSWERS

Within this measurable concept, the question was set to **"How profound are dynamic depths executed inside a DevSecOps CI pipeline?".** The answer to this question is based on the metrics for Dynamic Depth for applications for both phases. In phase one, It is immediately seen that the depth is given as one due to no implementation of dynamic tests within the CI pipeline. This depth is 40% lower than the required maturity level based on inherent risk. However, additions of OWASP ZAP to the CI pipeline were made in phase two. After several evaluations shown in the phase two section, it is now given that the depth in scan for applications is now set to three. This result shows a 40% improvement in security based on the MFAM metrics scale of 1 to 5. Thus, it is immediately seen that an improvement has occurred, which affected the maturity level.

STATIC DEPTH FOR APPLICATIONS QUESTION ANSWERS

The question here is, **"How profound is static analysis executed inside a Security DevOps CI pipeline?"**. This question can be answered based on the metrics and evaluations done for the Static Depth for applications in phases one and two. In phase one, the scans being done are being applied on the client or server-side components (e.g. backend/middleware). This made the depth be set as two, which is 20% lower than the required mark of three based on inherent risk. Additionally, in phase two, an addition of SonarCloud was made to the CI pipeline. This addition changed the maturity level to three, a 20% improvement from phase one

based on the MFAM metrics scale. This also shows that improvement has occurred due to adding static security tools to the CI pipeline.

INTENSITY QUESTION ANSWERS

How intense are tests being executed inside a DevSecOps CI pipeline? For example, what is being checked? The intensity is based on the evaluation given in phases one and two on Intensity. The initial intensity in phase one is set to two because the test had not been accustomed to only test what is being used. The required level based on inherited risk is set to three. However, phase one is 20% short of reaching that mark. There is a 20% improvement in phase two as the intensity level is now three. The maturity level could not go any higher because no heavyweight scans are being done. Finally, we can conclude that a 20% improvement has occurred in the CI Pipeline from the addition of SonarCloud and OWASP Zap Within the intensity axis.

CONSOLIDATION QUESTION ANSWERS

How comprehensive is the process of managing discoveries within a DevSecOps CI pipeline? The answer to this question is based on the metrics of consolidation in phases one and two. Based on the metrics reading of phase one. The completeness of findings is at level two because there is no specific way of handling false positives. There is a 20% improvement in the complete process of handling findings in phase two, bringing the maturity level to three. Thus, showing that improvements have occurred.

5.3.2 Improvement Objective From MFAM Model

The improvement objective for the MFAM model was set to be the improvement of security in the CI pipeline. The aim was to see how security by design can be measured in a CI pipeline when security tools are added. In the case study, we hoped to improve this by adding a SAST and DAST security tool to the CI pipeline of BiZZdesign. When we look at the questions asked within the different information axes, it can be concluded that security improved by adding certain checks in the CI pipeline. This statement means that the improvement objective was achieved. However, many notes are made, and improvements could still be made to aid future implementation. Some of these notes are mentioned within the case study and finally summarised in section 6.

From the introduction, it is stated that RQ1.1 is answered through this case study. It was also stated that a previous model (GQM) used to measure security improvements did not quantify how much was improved. However, the MFAM model showed that security improvements when security checks and tools are added to the CI pipeline are measurable. Therefore, the consensus is that these security additions improve the security of applications made at BiZZdesign. It is also important to note that the degree to which the improvement occurred at BiZZdesign can be quantified through the maturity level indicators.

5.4 Impact On Agility

The introduction of this thesis document discussed that the security team is looking for robustness, and developers are looking for agility. It is then seen as a crucial aspect to add to this thesis.

5.4.1 Impact of security tool additions on CI Pipeline

So, first, the impact on productivity when SonarCloud is added is discussed. Next, the impact when OWASP Zap is added is discussed and finally, when both tools are added. As it is already known, CircleCI is used for testing and building jobs at BiZZdesign. Meaning the impact on agility directly affects the CircleCI jobs. Fortunately, CircleCI has this feature called insightful which gives statistical data of build and test jobs. CircleCI also has concurrency in the job run, meaning five jobs can run simultaneously within a build. Before applying SonarCloud, the time for jobs to run took approximately 30 to 45 minutes in one area (Area One) of the system. It took approximately 1hrs31mins to 1hr50 mins in another area (Area Two) running a different code language.

SONARCLOUD IMPACT ON AGILITY

SonarCloud was added in about two areas of the whole system, one area with a different type of coding language and the other with another. Adding SonarCloud with default settings and no prior custom configuration to Area One increased the build time from 30 to 45 minutes to approximately 1:13:00 hrs. The time increases by approximately 28 minutes, which is almost half an hour. This is just the time it takes SonarScanner to send the gathered code to the SonarCloud server for analysis and not the time taken to run the analysis on SonarCloud Server. After some configurations were made with SonarScanner and SonarCloud, the time taken to run Area One was reduced to approximately 1hr 5mins. This reduction is approximately 7mins, thus improving the time to build after configuration.

In Area Two, the addition was configured before adding it to the build system. The time taken was approximately 3 hrs 2mins, which is about 1hr 20mins increase from the current build time. With success, several attempts were made to reduce the time of the SonarScanner build Job. Using the multithreading feature provided by SonarScanner, it was then possible to bring down the time of the build job to approximately 2hrs 20 mins which is a 42-minute reduction.

OWASP ZAP IMPACT ON AGILITY

OWASP Zap is instead run on Area One from the previous section on SonarCloud Impact on agility. The time taken by OWASP Zap did not impact the overall time of the build. Time taken for a build to run was still approximately 30 to 45 minutes. The reason for this is the feature in CircleCI that allows jobs to run simultaneously. This enabled ZAP to finish running its job before the other jobs were done.

5.4.2 Effects On Productivity

In general, Only SonarCloud had a significant impact on agility. This is because SonarScanner takes a significant amount of time to run on both areas it was applied. Understanding what this means to productivity, An interview was conducted with the lead developer at BiZZdesign. The primary outcome of the interview was the developer saying,

“Sonar takes a long time to run, sometimes my build has probably finished running before sonar lets me know that there is a bug here or there. By then, developers won't take the tool seriously as they have already pushed or merged their code. This makes sonar a tool which might just be making noise without a high impact.” This shows that adding this security is not enough. It is also important to configure it in a way that its impact on agility is reduced. Otherwise, it may reduce the usefulness of the tool when it comes to culture. It was already seen that attempts were made to achieve this with SonarCloud. There was a success in these attempts, but the time it took was still quite lengthy.

6 Conclusion

This chapter concludes the topics from all the chapters presented in this thesis. The main research question *is **"How much will security be improved when adding security tools into a CI pipeline?"***. The answer is covered throughout this report. First, we began with a systematic review of existing approaches on possible ways of adding security checks to a CI pipeline. It was then decided to limit the scope of the thesis to only the additions of DAST and SAST security checks.

6.1 A Model To Measure Security Improvements By Design

To help answer the main research question, another sub-research question was developed. The question RQ1.1 is *"How do we measure security improvements of a CI pipeline when evaluating security design tools or measures added to it?"*. The answer to this question leads to A Model For Measuring Improvement Of Security by design In CI pipelines. This model was named The Metrics and Four-axis Maturity Driven DevSecOps (MFAM). This model is based on an improvement objective. This improvement objective then leads to the information axis, which is the security of the areas within the CI pipeline that matter when adding security checks. This axis leads to measurable concepts that can be measured within a CI pipeline to produce better applications. This Measurable concept then leads to questions that need to be answered to help us reach our improvement objective. The questions are then answered with the help of our maturity metrics and supporting metrics. The metrics are then validated through OWASP SAMM and ISO27001 2017. These compliance and standard bodies did not give the metrics exactly but gave the standards that should be reached with an organisations applications security.

6.2 Validation Of Model With BiZZdesign

To validate the model. We checked to see if the model could work in practice. To achieve this goal, we took BiZZdesign as a case study. This case study then answers RQ1.2. ***"In practice, is it possible to measure security improvements when security tools are added to a CI pipeline?"***. The answer to this question was achieved through the different axis, i.e. static axis, dynamic axis, intensity and consolidation.

How much security will be improved when adding security tools into CI pipelines is arguable because of the complicated nature of Application Development. There are so many variables and instances to be considered. The MFAM model, however, has been able to quantify how much improvement was made. It is seen that based on the case study used, security by design was improved by around twenty to forty per cent after security tools were added. This percentage increase was based on the maturity scale of 1 to 5 and the axis presented in the MFAM model. This showed how much security by design improved when adding security checks to the CI pipeline. Thus, the goal of this thesis has been achieved. Never there are some critical takeaways learnt from this Engineering research.

6.3 Key Takeaways and Recommendations

1. Security can be measured when given the correct variables
2. Please do not underestimate how difficult it is to configure and install tools. i.e. complicated systems need simple integration mechanisms
3. Some of the severity level indicators of tools are not as accurate as they should be
4. Developers need specific policies to be enforced to make additions more useful
5. The time taken for tools to run have a direct impact on productivity, agility and tool usefulness
6. A Better method for fixing false positives and identifying false negatives is needed
7. A vulnerability management system is vital for a better understanding of security tools and the removal of defect duplicates.

6.4 Future Works

In this chapter, A look at possible future research needs to be looked at in several areas. In the case study used for this research, only one organisation had the application of the solution developed. This implies that the data available for this research is limited. Different companies have different types of continuous integration pipelines as well as DevOps systems. This statement means the results of measurements might be different or could show where improvements might be needed to the MFAM model. This way, one can validate that the MFAM model is more meaningful in practice and can be applied to all CI pipelines.

6.4.1 Applying Security Into The Development Culture

During this research, there were several observations and key takeaways notable that need further research. One major observation is in relation to Applying Security to the development culture. It was seen that even when security tools are added into the development pipelines, it does not necessarily change the culture unless enforced. This means that an organisation might integrate certain tools and checks just for the face of saying they are certified according to a standard. A security tool can be added, but yet defects in them are not looked into due to the culture of a set organisation. This reason shows that future research needs to be done in Applying Security to the development Culture, not just the cycle.

6.4.2 Improvement Of SAST Scanners

During the course of this thesis, there were several drawbacks seen in SAST tools. One of which is SAST takes time to run. The effects of this are seen in agility in the section on Impact On agility. In the early parts of this thesis, it was stated that agility is vital in the development cycle. Research on how to optimise SAST scanning to improve its agility is a topic that is quite important to cover in the future. The question is, how can we improve the agility of SAST tools?

6.4.3 Automated Defect Fix

Automation is a topic many today strive to achieve. Unfortunately, it is more easily said than done for complicated systems. In this thesis, it is seen that agility and false positives can

be reasons why developers do not fix defects in time. One way of solving this problem is through automating fixes of most of these defects. This is a research topic which is complicated yet needs to be looked at more in future. A solution can be having a certain tool that understands the overall structure of the development cycle, which in turn can apply automated fixes without the need for developers to do so.

BIBLIOGRAPHY

[1] International Organization for Standardization (ISO), International Electrotechnical Commission (IEC) and, Institute of Electrical and Electronics Engineers (IEEE), ISO/IEC/IEEE 16085:2021 Systems and Software Engineering — Life Cycle Processes — Risk Management. 2021.

[2] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What Is DevOps? a Systematic Mapping Study on Definitions and Practices," Proceedings of the Scientific Workshop Proceedings of XP2016 on - XP '16 Workshops, vol. 12, pp. 1–11, May 2016.

[3] N. Subramanian and A. Jeyaraj, "Recent Security Challenges in Cloud Computing," Computers & Electrical Engineering, vol. 71, no. 0045-7906, pp. 28–42, Oct. 2018.

[4] Red Hat, "What is DevSecOps?," Redhat.com, 2019. https://www.redhat.com/en/topics/devops/what-is-devsecops (accessed May 11, 2021).

[5] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," Journal of Systems and Software, vol. 123, pp. 176–189, Jan. 2017.

[6] Help Net Security, "53% of enterprises have no idea if their security tools are working," Help Net Security, Jul. 31, 2019. https://www.helpnetsecurity.com/2019/07/31/are-security-tools-working/ (accessed May 11, 2021).

[7] A. Tan and TechTarget, "Security remains an afterthought in DevOps," ComputerWeekly.com, Mar. 06, 2018. https://www.computerweekly.com/news/252436268/Security-remains-an-afterthought-in-DevOps (accessed May 15, 2021).

[8] Synopsys, "What Is DevOps and How Does It Work? | Synopsys," www.synopsys.com, Jun. 11, 2021. https://www.synopsys.com/glossary/what-is-devops.html (accessed Jul. 17, 2021).

[9] A. Hevner, S. March, J. Park, and S. Ram, "Design Science in Information Systems Research," MIS Quarterly, vol. 28, no. 1, p. 75, Mar. 2004.

[10] P. J. Denning and R. M. Metcalfe, Beyond Calculation: The Next Fifty Years of Computing, 1st ed. New York: Copernicus Books, 1997, pp. 259–265.

[11] A. Hevner, "A Three Cycle View of Design Science Research," Scandinavian Journal of Information Systems, vol. 19, no. 2, pp. 1–6, Jan. 2007.

[12] S. Kumar and P. Dubey, "Software Development Life Cycle (SDLC) - Analytical Comparison and Survey on Traditional and Agile Methodology," Yumpu, vol. 2, no. 8, pp. 22–30, Aug. 2013, Accessed: May 17, 2021. [Online]. Available: https://www.yumpu.com/en/document/view/28065623/software-development-life-cycle-sdlc-abhinav-institute-of-.

[13] S. Buehring, What is DevOps? | Free PDF ebook, vol. 2. London, England, UK.: Knowledge Train, 2021.

[14] E. Mueller, “What Is DevOps?,” The agile admin, Feb. 02, 2010. https://theagileadmin.com/what-is-devops/ (accessed Jun. 16, 2021).

[15] NetApp, “What Is DevOps? - Practices and Benefits Explained,” netapp.com, 2019. https://www.netapp.com/devops-solutions/what-is-devops/ (accessed Jun. 08, 2021).

[16] R. Kumar and R. Goyal, “Modeling Continuous security: a Conceptual Model for Automated DevSecOps Using open-source Software over Cloud (ADOC),” Computers & Security, vol. 97, no. 101967, Oct. 2020, Accessed: Jun. 10, 2021. [Online].

[17] B. Potter and G. McGraw, “Software security testing,” IEEE Security & Privacy Magazine, vol. 2, no. 5, pp. 81–85, Sep. 2004.

[18] M. Felderer and E. Fourneret, “A systematic classification of security regression testing approaches,” International Journal on Software Tools for Technology Transfer, vol. 17, no. 3, pp. 305–319, Jan. 2015.

[19] Kaspersky, “What Is Zero Day Exploit? - Definition and Explanation,” www.kaspersky.com, Feb. 27, 2018. https://www.kaspersky.com/resource-center/definitions/zero-day-exploit (accessed Jun. 22, 2021).

[20] B. Lundgren and N. Möller, “Defining Information Security,” Science and Engineering Ethics, vol. 25, no. 2, pp. 419–441, Nov. 2017.

[21] Open Web Application Security Project® (OWASP), “Attacks on Software Application Security | OWASP Foundation,” owasp.org, 2000. https://owasp.org/www-community/attacks/ (accessed Sep. 07, 2021).

[22] Puppet and Splunk, “State Of DevOps Report,” Puppet, Portland, United States, Nov. 2019. Accessed: Jun. 13, 2021. [Online]. Available: https://media.webteam.puppet.com/uploads/2019/11/Puppet-State-of-DevOps-Report-2018_update.pdf.

[23] IBM Cloud Education, “DevSecOps - What Is DevSecOps?,” IBM, Jul. 2020. Accessed: May 22, 2021. [Online]. Available: https://www.ibm.com/cloud/learn/devsecops.

[24] Puppet, CircleCi, and Splunk, “2019 State of DevOps Report,” puppet, Nov. 2020. Accessed: Jul. 27, 2021. [Online]. Available: https://puppet.com/resources/report/2019-state-of-devops-report/.

[25] S. Danielson, EE. Kathryn, D. Coulter, T. Petersen, and M. Jacobs, “Add Continuous Security Validation to your CICD Pipeline - Azure DevOps,” docs.microsoft.com. Apr. 26, 2018, Accessed: Jun. 21, 2021. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/migrate/security-validation-cicd-pipeline?view=azure-devops&viewFallbackFrom=%20azure-devops.

[26] S. Koussa, “What Do SAST, DAST, IAST and RASP Mean to developers?,” Softwaresecured.com, Nov. 02, 2018. https://www.softwaresecured.com/what-do-sast-dast-iast-and-rasp-mean-to-developers/ (accessed Jul. 16, 2021).

[27] Hdiv security, “IAST - Interactive Application Security Testing,” Hdiv Security, Gipuzkoa, Spain, 2017. Accessed: Jul. 12, 2021. [Online]. Available: https://hdivsecurity.com/downloads/94hfsvq9q3vtc23z862cr48XrT39Bze6/Hdiv--What-Is-IAST.pdf.

[28] Open Web Application Security Project (OWASP), “Static Code Analysis,” Owasp.org, 2013. https://owasp.org/www-community/controls/Static_Code_Analysis (accessed Jul. 19, 2021).

[29] Open Web Application Security Project® (OWASP), “Source Code Analysis Tools | OWASP,” owasp.org, 2017. https://owasp.org/www-community/Source_Code_Analysis_Tools (accessed Aug. 05, 2021).

[30] Open Web Application Security Project® (OWASP), “Vulnerability Scanning Tools | OWASP,” owasp.org, 2017. https://owasp.org/www-community/Vulnerability_Scanning_Tools (accessed Aug. 15, 2021).

[31] J. Peterson, “IAST: Interactive Application Security Testing,” WhiteSource, Jul. 16, 2020. https://www.whitesourcesoftware.com/resources/blog/iast-interactive-application-security-testing/ (accessed Aug. 12, 2021).

[32] D. Wichers, D. Wetter, and S. Koussa, “Free for Open Source Application Security Tools | OWASP,” owasp.org, 2017. https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools (accessed Aug. 19, 2021).

[33] P. Rooney, “Study: More than 50% of Global 500 Use Vulnerable Open Source Components,” ZDNet, Scituate, Massachusetts, USA, Mar. 2012. Accessed: Aug. 08, 2021. [Online]. Available: https://www.zdnet.com/article/study-more-than-50-of-global-500-use-vulnerable-open-source-components/.

[34] Black Duck by Synopsys, “2015 Future of open-source Survey Results,” slideshare.net, Norway, Apr. 2015. Accessed: Aug. 14, 2021. [Online]. Available: https://www.slideshare.net/blackducksoftware/2015-future-of-open-source-survey-results/24-CASE_STUDYCUSTOMERCASE_STUDY_FORBast_Fosen.

[35] GitLab, “Dependency Scanning | GitLab,” docs.gitlab.com, 2020. https://docs.gitlab.com/ee/user/application_security/dependency_scanning/ (accessed Aug. 09, 2021).

[36] Trend Micro, “What Is Container Security?,” Trend Micro, Sep. 08, 2021. https://www.trendmicro.com/en_us/what-is/container-security.html (accessed Sep. 09, 2021).

[37] P. Runeson, “A survey of unit testing practices,” IEEE Software, vol. 23, no. 4, pp. 22–29, Jul. 2006, Accessed: Apr. 29, 2020. [Online].

[38] Think Tech Advisors, “What is IT Maturity? | Think Tech Advisors,” thinktechadvisors.com, Mar. 2017. https://thinktechadvisors.com/2017/03/what-is-it-maturity/#:~:text=IT%5C%20maturity%5C%20is%5C%20a%5C%20measure (accessed Sep. 27, 2021).

[39] BiZZdesign, “Our mission and approach,” BiZZdesign, 2020. https://bizzdesign.com/why-bizzdesign/ (accessed Aug. 13, 2021).

[40] A. Mullans, “Keep all your packages up to date with Dependabot,” The GitHub Blog, Jun. 01, 2020. https://github.blog/2020-06-01-keep-all-your-packages-up-to-date-with-dependabot/#:~:text=Dependabot%5C%20alleviates%5C%20that%5C%20pain%5C%20by (accessed Sep. 28, 2021).

[41] M. Kandut, “What is NPM audit?,” www.mariokandut.com, Feb. 06, 2021. https://www.mariokandut.com/what-is-npm-audit-and-how-to-increase-security/.

[42] NPM Docs, “npm-audit | npm Docs,” docs.npmjs.com, 2014. https://docs.npmjs.com/cli/v7/commands/npm-audit (accessed Aug. 11, 2021).

[43] M. Z. Neto G. A. A. Santana, F. Sapata, M. Munoz, A. M. S. P. Moraes, T. Morais, D. L. Goldfarb, AWS Certified Security Study Guide: Specialty (SCS-C01) Exam. John Wiley & Sons, 2021.

[44] Travis CI, “Travis CI Documentation,” docs.travis-ci.com, 2021. https://docs.travis-ci.com/user/sonarcloud/ (accessed Aug. 07, 2021).

[45] Open Web Application Security Project® (OWASP), “OWASP ZAP – Getting Started,” www.zaproxy.org, 2017. https://www.zaproxy.org/getting-started/ (accessed Sep. 01, 2021).