INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




An Approach Toward Implementing Continuous Security In Agile Environment

DOAA ASSAF
Julho de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

An Approach Toward Implementing Continuous Security In Agile Environment

Doaa Assaf

Dissertation to obtain the master's degree in Computer Engineering, Specialization Area in Computer and system engineering

Supervisor: Dr. Jorge Pinto Leite

Jury:

President:

[President's Name, Category, School]

Vowels:

[Name of vowel1, Category, School]

[Name of vowel2, Category, School]

Porto, July 2020

Dedication

To my parents, Baher and Ibtisam,

For their endless love and support.

To all the people who believed in me,

To Portugal, my new beloved home.

Abstract

Traditionally, developers design software to accomplish a set of functions and then later add—or do not add—security measures, especially after the prevalence of the agile software development model. Consequently, there is an increased risk of security vulnerabilities that are introduced into the software in various stages of development. To avoid security vulnerabilities, there are many secure software development efforts in the directions of secure software development lifecycle process.
The purpose of this thesis is to propose a software security assurance methodology and integrate it into the Msg Life organization's development lifecycle based on security best practices that fulfill their needs in building secure software applications.
Ultimately, the objective adhered to increasing the security maturity level according to the suggested security assurance roadmap and implemented partly in the context of this thesis.

Keywords: software security assurance, SSDLC, information security, secure development lifecycle, SAMM project, DevSecOps.

Resumo

Tradicionalmente, os desenvolvedores projetam o software para realizar um conjunto de funções e, posteriormente, adicionam - ou não - medidas de segurança, especialmente após a prevalência do modelo de desenvolvimento ágil de software. Consequentemente, há um risco aumentado de vulnerabilidades de segurança que são introduzidas no software em vários estágios de desenvolvimento. Para evitar vulnerabilidades de segurança, existem muitos esforços no desenvolvimento de software nas direções dos processos do ciclo de vida desse mesmo software. O objetivo desta tese é propor uma metodologia de garantia de segurança de software e integrá-la ao ciclo de vida de desenvolvimento da Msg Life Company, com base nas melhores práticas de segurança que atendem às suas necessidades na criação de aplicativos de software seguros.

Por fim, o objetivo aderiu ao aumento do nível de maturidade da segurança de acordo com o roteiro sugerido de garantia de segurança e implementado parcialmente no contexto desta tese.

Palavras-chave: Garantia de segurança de software, SSDLC, ciclo de vida de desenvolvimento seguro, projeto SAMM, DevSecOps.

Thanks

I am sincerely grateful to the Global Platform of Syrian Students to give me the opportunity to be here today. Special thanks to all the great people who are working behind it, including the former president of Portugal Jorge Sampaio and Dr. Helena Barocco.

Thanks to Professor Jorge Pinto Leite my supervisor for his invaluable guidance.

Thanks to Professor Antonio Costa for his assistance throughout the early stages of this thesis.

Thanks to Msg Life company for being such a great working environment, special thanks to my external supervisor Nuno Alves.

Thanks to my beloved family for pushing me to be the best in all of my endeavors.

Thanks to my friends who has been always there for me, and I would like to offer my special thanks to my bestie HSN, for reviewing this thesis and being my first supporter along the way.

Index

List of Figures

List of tables

List of source code

Acronyms

API: Application Programming Interface.
BOM: Bill-Of-Material.
BSIMM: Building Security in Maturity Model.
CI/CD: Continuous Integration and Continuous Delivery.
CRUD: create, read, update, delete operations
DevOps: Development and Operations.
DevSecOps: Development, Security, and Operations.
GDPR: General Data Protection Regulation.
HIPAA: Health Insurance Portability and Accountability Act.
MS-SDL: Microsoft Security Development Lifecycle.
OWASP CLASP: Comprehensive, Lightweight Application Security Process.
OWASP: Open Web Application Security Project.
PCI: payment card industry.
RestApi: Representational state transfer Application Programming Interface.
SAMM: Software Assurance Maturity Model.
SDLC: software development lifecycle.
SSDLC: Secure software development lifecycle
ZAP: Zed attack proxy.

.

1 Introduction

The essence of this thesis is a practical solution for securing the development lifecycle through one of the robust security approaches. In this chapter, we bottom line the objective, the problem, concluding with the document structure.

1.1 Background and context

Cybersecurity is the second-highest concern globally in 2019 and the highest risk facing Europe for doing business, also, data fraud or theft comes the 7th globally according to world economic forum. (Schwab, 2019) Despite that fact, many companies don't take security seriously and try to avoid dealing with it as much as possible. (Pal and Pantaleo, 2005)

Thus, the security practices in many cases are considered an extra thing to do in the testing phase into the development lifecycle. This inconsistent relationship was not clear before when companies adapted traditional Software Development Lifecycle (SDLC) like a waterfall or incremental model. The reason for it was because these models by nature enforce rich documentation, no changes in the requirements and they just need to do security functions in a predefined order. The issue raised after the agile manifesto in 2001[1] which requires special behaviors and techniques that can't get along easily with security practices inside the companies. (Siponen, Baskerville and Kuivalainen, 2014)

That enforces many organizations and experts in both fields to put agile model and security practices on the same page and find the appropriate level of combination between them. Moreover, this is where Secure Software Development Lifecycle (SSDLC) comes into the picture to help security involves in the software development lifecycle smoothly in a cost-efficient- manner. In 2004 Microsoft releases its SDL[2], it followed up by OWASP CLASP the

[1] http://www.agilemanifesto.org/principles.html
[2] https://www.microsoft.com/en-us/securityengineering/sdl

assurance process from OWASP[3] and Gary McGraw Touchpoint[4] 7 principles, SEI Team Software Process for Secure Software Development, SAMM project from OWASP[5] also, and many more. Each one of them has its own pros and cons in a specific working environment and companies must choose the most appropriate one to adapt based on many factors.

1.2 Problem

The IT infrastructure landscape has subjected to exponential changes over the past decade. The shift to agile, DevOps and dynamic applications has brought huge benefits to organizations looking to rapid growth using advanced applications and services that fulfill the continuous integration and continuous delivery concepts. (Kaur, Jajoo and Manisha, 2015)

However, *"while the applications have stormed ahead in terms of speed, scale, and functionality, they are often lacking in robust security and compliance".* (*What is DevSecOps? Defined, Explained, and Explored | Forcepoint*, no date), mainly because of the uneasy relationship between agile and security.

Agile development emphasizes flexibility and rapid changes, while security methodologies rely on a more systematic approach to development. (Mougouei *et al.*, 2013)

Going back to agile manifesto, where this can be clarified in more detail. According to the manifesto, agile highest priority is to match up to customer needs through early and continuous delivery, which raises an issue where security will be tolerated from poorly security-aware customers. (Goertzel and Hamilton, 2007)

Also, changing in the requirements, has a negative impact on security, because each change in the requirements means changing the design and the architecture in the system, that imposes changing the system's threat model, attack surface, and introduce new risks. (Goertzel and Hamilton, 2007)

Agile assumes that developers are trusted to achieve the work, but that is dangerous from a security perspective unless the developers are security-aware, comply
to secure programming, and security is integrated into their process. (Goertzel and Hamilton, 2007)

Face to face communication is preferred over documentation, this also cannot get along easily with security, because security by principle needs to be based on actual official evidence and documentation. (Goertzel and Hamilton, 2007)

Agile identifies the measure of success by producing working software, while in security it is the production of the best secure application. (Goertzel and Hamilton, 2007)

Thus, achieving the right balance between these two - equally vital- aspects of the development process is never easy. (Goertzel and Hamilton, 2007)

[3] https://owasp.org/
[4] http://www.swsec.com/
[5] https://owasp.org/www-project-samm/

Msg Life is a software organization that provides insurance software products to its customers worldwide has been using agile development approach.

The organization has been working on providing solutions and services to its customers as soon as possible. So, they implement security in an ad-hoc approach, these services are considered risky in terms of the personal data they handle, and this sort of applications is among the highest targets to attackers globally and could affect the organizations for years according to many reports in the last few years. (Ibm, 2019)

Hence, the organization wants to improve its security approach and improve the security of its products, which arise the integration between its security practices and the development process as an important issue.

1.3 Objective and proposed approach

Msg Life Company's main objective is to evaluate its current security situation and improve its security plan.

This thesis aims to ensure that Msg Life implements an appropriate level of secure software by fixing the current security vulnerabilities, use the best available security approach from the very beginning of the development lifecycle and shift-left security activities[6], guide secure software activities, set the ideal behaviors and practices that enable teams to get maximum value from Agile development, as well as from the security techniques.

which by result mitigates most common vulnerabilities earlier, and reduce the cost of flaws in the system in the future.

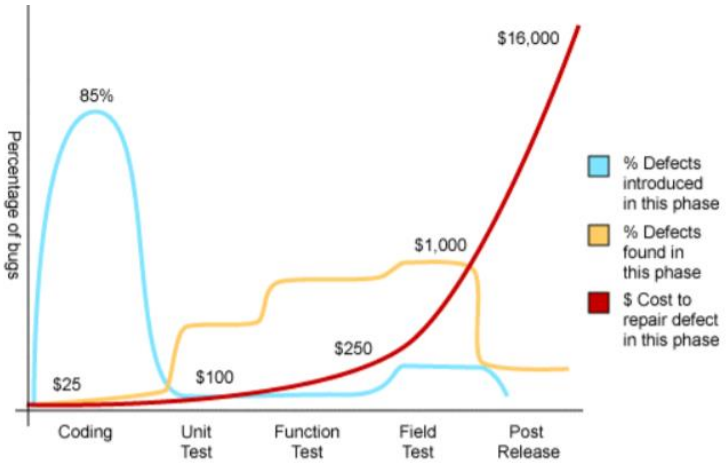

Figure 1 - Cost to fix defects in different phases into the SDLC without applying shift-left (Hermiyanty, Wandira Ayu Bertin, 2017)

[6] Shift-left is a software development concept refers to moving the activities to an early stage in the development lifecycle.

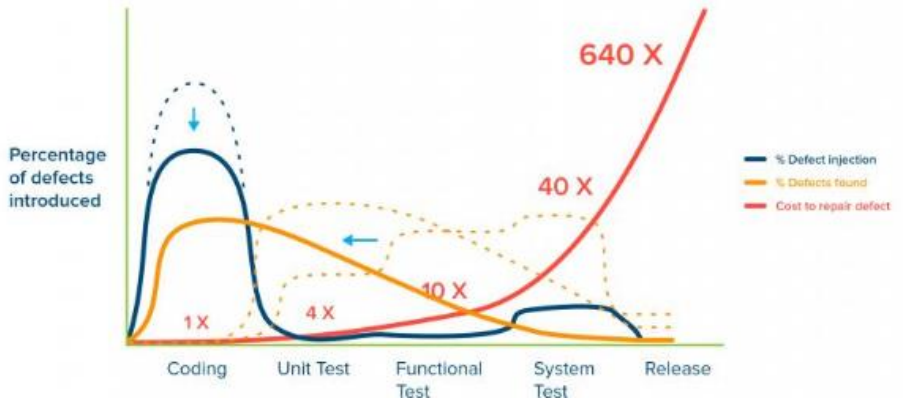

Figure 2 - fixing defects cost in different phases into the SDLC after applying shift-left (Hermiyanty, Wandira Ayu Bertin, 2017)

To achieve that goal the following approach is proposed:

- Evaluating the current situation and the security maturity level in the company.
- Identifying the objectives and the exact level that we need to reach in the company
- Setting the appropriate Secure Software Development Lifecycle roadmap based on the previous maturity evaluation, with respect of the development model in the company and general working environment.
- Define strategic and tactical tracks for each objective.
- Implementing the previous approach.
- Evaluating the results and compare them with the objectives.

1.4 Expected results

With the development of this project, it is expected that the solution will allow the following:

- More secure software as security is a *continuous* concern, through a repeatable and measurable process and integrating it with foundational software development activities.
- Reducing or preventing damage caused by attacks.
- Early detection of flaws in the system.
- Provide secure software development reviews.
- Reducing the costs of repairing security weaknesses in applications.
- Significantly decreasing the number of vulnerabilities in the application when it is ready to go live, thereby reducing delays in the go-live process.
- Awareness of security considerations by developers and stakeholders.
- Overall reduction of inherent business risks for the organization.

1.5 Outline

As this thesis will show, the first chapter, 1 Introduction, where the main problem is defined along with the objectives and the expected results.

In the second chapter, **State-of-the-art**, some of the existing approaches to solve the problem are discussed, including some well-known secure development lifecycle models and technologies used in this field to support those approaches.

In the third chapter, **Value Analysis**, a study about the value of this project is included, and the NCD model, with the QFD representation.

The detailed design of the selected approach in Msg Life will be clarified in the fourth chapter Design the SAMM project solution for Msg Life. While the fifth chapter, Implementing the solution, the largest chapter in this thesis, includes the organization's security roadmap and its phases based on the selected approach, besides, the implemented part of this roadmap in the context of this thesis.

In the sixth chapter, **Experimentation and Evaluation of the Solution** presents the factors that are used to evaluate the outcomes, including test hypotheses, the evaluation methodologies, and the statistic tests to evaluate each factor.

Finally, in the **Conclusion** chapter, the summary and future work are covered.

2 State-of-the-art

In the chapter, the State-of-the-Art is presented, where the concepts and methodologies related to the problem are outlined, side by side, with the underlining technologies that assert these approaches in the Technological Background.

2.1 Secure software definition

It is the immune software that cannot be compromised under malicious attacks and continuously function correctly and predictably. (Goertzel and Hamilton, 2007)

Many underlying practices throughout the development must be ensured to produce a secure software application.

To describe software as a secure software it must avoid, by all meanings, the exploitable vulnerabilities, and flows, including the malicious code and faults that are implemented intentionally by developers.

Moreover, the connections between the components of the software are also secured, as well as the communication between the software and external entities. (Goertzel and Hamilton, 2007)

2.2 Software security assurance definition

According to Committee on National Security Systems (CNSS) definition, it is the software trustworthiness by reaching and ensuring an acceptable and reasonable level of justifiable confidence that the developing software will function correctly and predictably, and this software cannot be compromised by direct attack or malicious code. (Jarzombek, 2012)

2.3 SSDLC definition

Secure Software Development lifecycle is a process model used by organizations to build secure applications. The SSDLC Process defines how to integrate security assurance activities such as design review, architecture analysis, code review, and penetration testing into the development lifecycle. (Goertzel and Hamilton, 2007)

There are established methodologies that cover different software development models like waterfall, incremental, spiral, and agile that organizations use and models they follow to address different challenges and goals.

In the following section, secure software methodologies are disclosed in detail.

2.4 Existing SSDLC methodologies

Some SSDLC methodologies have been developed and proposed, the main ones are summarized in this section.

2.4.1 Microsoft SDL

Microsoft SDL is the first software security assurance process that is focused on software development. Microsoft has adopted SDL as a mandatory policy in 2004 in order to resolve the security concerns that have previously arisen in its products. The Microsoft SDL model consists of several security activities. The activities are categorized into mandatory and optional activities and are grouped by the phases of the software development lifecycle (SDLC), as shown in Figure 3. (Davis, 2013)

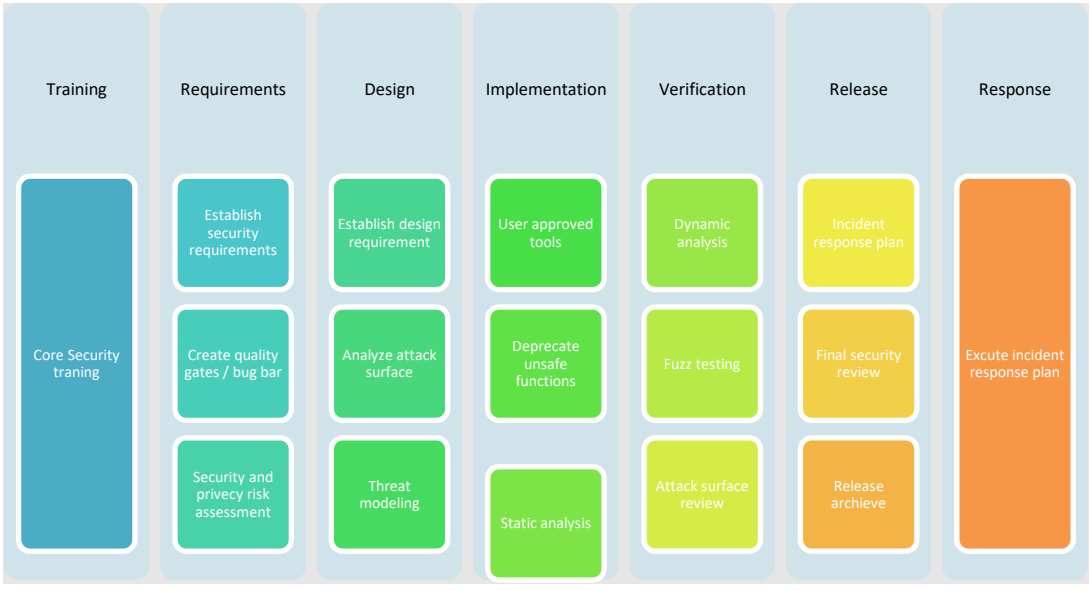

Figure 3 - Microsoft SDL mandatory practices in each development phase (Microsoft Corporation, 2010)

- Microsoft has two types of activities in its SDL, the mandatory activities, where the software must pass the following security phases(Davis, 2013):

- **Pre-SDL requirement:** Core security training to ensure that everyone understands the attacker's perspective, their goals because the security is everyone's responsibility.
- **Requirements phase:** the security team gather the security requirements. Also, establish a quality gate and bug bar to enforce a quality policy in the organization and classifies vulnerabilities based on their effects from a list of STRIDE values: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), and Elevation of Privilege (EoP).
- **Design phase:** Identifies the design requirements, through an attack surface analysis which is basically gathering all the different points where an attacker could get into a system, and where they could get data out (User interface (UI) forms and fields, HTTP protocol headers and cookies, APIs, Files, Databases, Other local storage). And the core of this phase is the threat modeling where the development team consider, document, and discuss the security implications of designs in the context of their planned operational environment in a structured manner.
- **Implementation phase:** The development team decides and approves the set of tools that will be used during the development process. And perform static code analysis and analyzing the source code to ensure that secure coding policies are being followed.
- **Verification phase:** Dynamic tests for the full software to check against security and privacy specifications. It can be easily integrated with CI/CD tools.
- **Release phase:** the development team creates an incident response plan that identifies the roles and activates in case of an emergency. The team must also perform a Final Security Review.

The other set of activities are the optional ones for high critical software to enrich the software assurance plan:

- **Manual Code Review:** Manual code review of the software or parts of it.
- **Penetration Testing:** perform attack scenarios that aim to unveil potential security flaws and vulnerabilities in the system.
- **Vulnerability Analysis:** Investigating similar Applications that provide similar functionalities or use similar technologies, vulnerabilities databases that could help in avoiding potential security issues during the design and implementation.

2.4.2 Microsoft SDL for agile

Microsoft SDL works fine in big projects and the ones that rely on waterfall or incremental model. But after the agile revolution, Microsoft SDL was hard and even impossible to comply with it, in the form that was found in 2004. Mainly because all the SDL activities cannot be completed in each agile sprint.

In agile the requirements, architecture, and design evolve over time, so the threat model as an example becomes outdated quickly and the data sensitivity and connection to third parties may not be immediately known.

So, another version of SDL was released, it is called SDL/agile. *(SDL-Agile: Microsoft's Approach to Security for Agile Projects | Tech·Ed Europe 2009 | Channel* , 2009)

SDL for agile breaks the SDL into three categories of requirements:

- **Every-sprint requirements:** the requirements so important that they must be completed every iteration, the green activities in Figure 4.
- **One-time requirements:** the requirements that only must be completed once per project no matter how long it runs, the blue activities in figure 4.
- **Bucket requirements:** the requirements that still need to be completed regularly but are not so important that they need to be completed every sprint, the orange activities in Figure 4.

Figure 4 - Microsoft SDL activities group based on SDL/agile (Microsoft Corporation, 2010)

2.4.3 Touchpoint

It is a set of best practices, it was introduced in Part II of Gary McGraw's -Software Security: Building Security In- book. These best practices first appeared as a set in 2004 in IEEE Security & Privacy magazine. Since then, they have been adopted by the U.S. government. (McGraw, 2006)

The touchpoints provide a mix of lightweight black-hat and white-hat activities. Some touchpoints are by their nature more effective than others, and the most effective should be adopted ones first. Touchpoints in order of effectiveness are:

1. Static analysis/Code review: entails the use of static analysis tools to detect common vulnerabilities.
2. Architectural risk analysis: identifying possible attacks and document these assumptions to uncover architectural flaws for mitigation.
3. Penetration testing: black-box testing for the application by using automated testing tools.
4. Risk-based security tests: risk-based security testing of the software based on attack patterns.
5. Considering abuse cases: an imaginary situation that describes the system behavior when it is under attack to identify the parts that need more protection.
6. Security requirements: specify functional security requirements and emergent ones that revealed from the abuse cases[7] and attack patterns.
7. Security operations: monitoring the application after deployment.

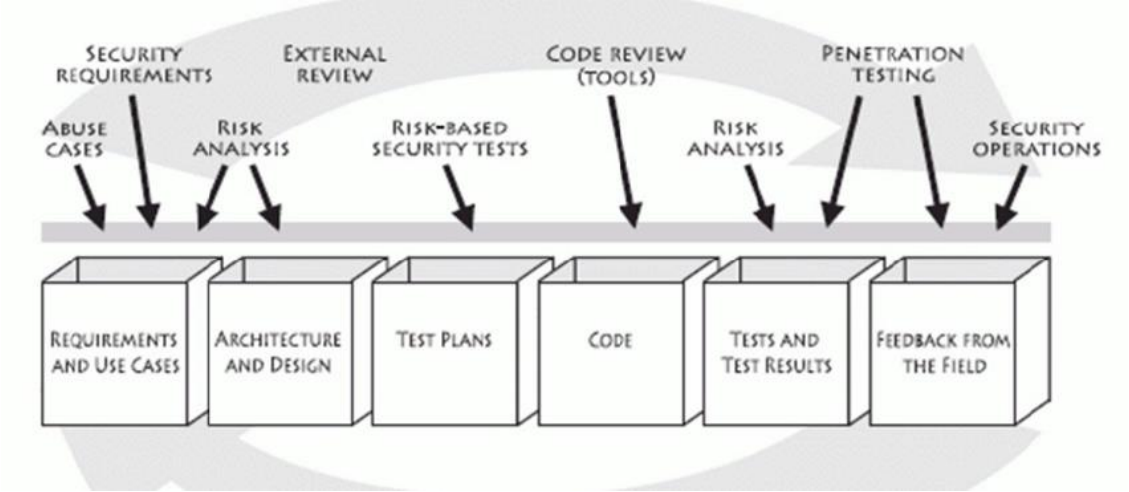

Figure 5 - Touchpoint artifacts in SDLC (Addison- Wesley, 2006).

Another bonus practice was mentioned as well: external analysis, which suggests outsourcing analysts to perform an independent security review or tests for the design and the implementation.

2.4.4 OWASP CLASP

OWASP released CLASP in 2006. It is a lightweight software security assurance process; it adopts an easy and effective approach for constructing secure software by introducing extensions to the traditional software engineering activities and providing implementation guidance in certain security areas. (De, Distrinet and Leuven, 2007)

[7] The abuse cases is the interaction between a system and one or more actors, when the consequence of the interaction damage the system.(Mcdermott and Fox, no date)

The CLASP process consists of 24 top-level activities that can be fully or partially incorporated into software that is being constructed. They can be grouped under 7 best practices according to the roles performed during development(*The CLASP Application Security Process The CLASP Application Security Process Introduction 1*, 2005):

1. Institute awareness programs
2. Perform application assessments
3. Capture security requirements
4. Implement secure development practices
5. Build vulnerability remediation procedures
6. Define and monitor metrics
7. Publish operational security guidelines

2.4.5 SAMM Project

OWASP released this open framework in 2008 for security assurance to help organizations formulate and implement a strategy for software security that is tailored to the specific risk facing the organization. It can be utilized by small, medium, and large organizations using any style of development waterfall or agile. Companies can determine which of the practices and sophistication levels are most appropriate for them. (Chandra, no date)

It maps from CLASP activities into SAMM's practices. Each objective in SAMM can be mapped to s existing standards (PCI, COBIT, ISO-17799/27002), by achieving this objective the corresponding parts of existing standards are achieved.

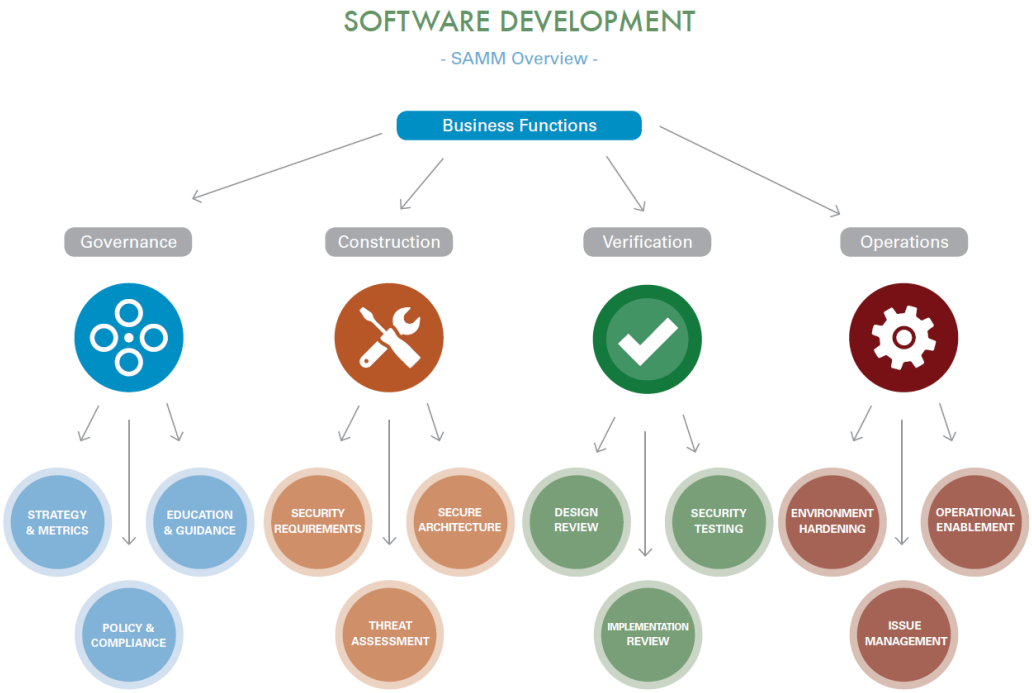

Figure 6 - SAMM project functions and practices

As we see in Figure 6, SAMM project divides the practices into four top functions (OWASP, 2017) :

- **Strategy & Metrics:** The way the organization looks at business risk and aligning security with it, like establishing a software security assurance program. For example, classifying data and application based on their business risk and security goals. This practice should be across the company.
- **Policy & Compliance**: understanding and meeting external legal requirements while driving internal security standards to ensure compliance. Like complying with PCI standard for online payment. The compliance must be clarified in the company and for each project.
- **Education & Guidance:** training and guidance on security topics. For developers and everyone involved in the development and business process, at a basic level it could be a generic training for developers but to reach an advanced maturity level, specific training for different stakeholders might be necessary.
- **Threat Assessment:** this practice contains the activities that help in identifying who would benefit from exploiting the system and prioritizing threats. It should be performed in an early phase of the development process, but it is never too late to do threat assessment in an advanced phase. Besides, it needs to be checked and updated if there are any changes in the architecture of the system.
- **Security Requirements:** focuses on identifying the functional and non-functional security requirements in the requirement definitions, for example, session time out is 30 minutes.
- **Secure Architecture:** imply secure design principles to the system design like secure by default principle, defense in depth, securing the weakest link.
- **Design Review:** asserting that software design does not include any security issues and it avoids security flaws. Analyzing the software attack surface is one of design review activities, that is done Based on threat model. The documents produced from the design review need to be updated if there are any changes in the system design each agile sprint.
- **Implementation Review:** reviewing the code and Analyzing it from common vulnerabilities, its activities could be as simple as having a checklist on things we need to make sure of, or review high-risk code manually, or using automated code analysis.
- **Security testing:** testing the fully implemented software in its run-time environment from known vulnerabilities.
- **Issue Management:** focusing on keeping the company ready to deal with any emergency incident issue in the software that the organization has developed.
- **Environment Hardening:** checking the operational environment health from necessary patches, having good documentation for the infrastructure, and updates for third party components.
- **Operational Enablement:** maintaining a good level of communication between the Development and Operation teams for critical security data depending on each

release, putting a change management procedure and code signing in an advanced level of the operational enablement.

2.5 Comparison between Existing SSDLC approaches

Comparing between MS-SDL, touchpoint, OWASP CLASP and SAMM project activities in different phases in the SDL is presented in Table 1 (Bedi, Gandotra and Singhal, 2013) (De, Distrinet and Leuven, 2007)(Nunez, Lindo and Rodriguez, 2020)

Table 1 - comparing between different SSDLCs activities

Development phase	**Microsoft SDL and Microsoft SDL for Agile**	**Touchpoint**	**OWASP CLASP**	**SAMM project**
Project inception	Assignment of a security advisor. Defining several roles (or teams). Define the bug bar. Define security-related fields to track security issues. Core security training	None.	Assignment of a security advisor. Identification of the metrics. Evaluate global requirement Security training	Policy and compliance Strategy and metrics Education and guidance
Requirement	Identify security objectives and security requirements.	Identify security requirements	Risk analysis. Threat Modeling. Identification of attackers and attack surface. Specification of misuse cases with their mitigations. Identification of Security requirements.	Risk analysis. Threat Modeling. Identification of attackers and attack surface. Specification of misuse cases with their mitigations. Identification of Security requirements.

Design and architecture	Product risk assessment. Identification of the security-critical components. Threat modeling Analyzing attack surface Identification of design techniques/guidelines	Risk analysis	Following design guidelines. Annotation of class diagrams with security information. Threat modeling. Risk analysis. Set cases of misuse	design review Threat modeling. Risk analysis. Analyzing attack surface.
Implementation	Following the secure coding standards. Security code reviews. Security testing. Using approved tools.	Using abuse cases and security requirements	Following secure coding guidelines. Code reviews. Use of static code analysis tools. Security testing	Following secure coding guidelines. Code reviews. Use of static code analysis tools. Security testing.
Testing	Dynamic analysis Fuzz testing	Security testing	Security testing	Security testing
Release and post-release	Writing of user manuals. Incident response management.	Monitoring after release.	Writing of user manuals. Incident response management. Code signing.	Writing of user manuals. Incident response management. Code signing.

2.6 Technical background: Methods, Techniques, and Tools for Secure Software development

There are already many tools and technologies to support security in different development phases. In this section, some of them are presented.

2.6.1 Risk rating methodologies

It is the process of identifying and estimating the severity associated with risks to the business, to allow eliminating them later. (Stoneburner, Goguen and Feringa, 2002)

- OWASP Risk Rating Methodology

It is one of the most known risk rating methods from OWASP. It uses the following simple equation to estimate risks:

$$Risk = Likelihood * Impact$$

Using this methodology, companies can estimate the severity of risks to business and make decisions on their top priority threats, to eliminate first. It also contains a rating system. So, taking this decision process is more straightforward. (*OWASP Risk Rating Methodology*, no date)

The process consists of six steps:

- *Identifying the risks:* the first step is knowing the risk.
- *Factors for Estimating Likelihood:* How likely to this threat to be uncovered and exploited by an attacker. There are many factors to estimate the likelihood. Some of them are attacker oriented like the skill level, motivation. Etc. while the other related to the risk itself like Ease of discovery.
- *Factors for Estimating Impact:* Each threat has a technical impact on the application. and business impact on the business.
- *Determining Severity of the Risk: From the previous step and the overall impact, the severity of the threats can be calculated.*
- *Deciding What to Fix.*
- *Customizing Your Risk Rating Model.*

- OCTAVE

It stands for Operationally Critical Threat, Asset, and Vulnerability Evaluation. It a risk assessment framework created by Carnegie Mellon University's Software Engineering Institute (SEI) in collaboration with CERT.

It has two distributions on for big organizations while the other one, which called OCTAVE-S for small businesses. And it targets organizational risks more than technical ones. (Alberts *et al.*, 1999)

OCTAVE suffers from some main issues like being complex and it does not offer activities and practices to mitigate web application risks. (Jagannathan, no date)

2.6.2 Analyzing attack surface

Attack Surface Analysis is about mapping out what parts of a system need to be reviewed and tested for security vulnerabilities and understand the risky areas in the application. (Attack Surface Analysis Cheat Sheet | OWASP, no date)

In other words, attack surface analysis identifies the exit and entry points, which are the systems' methods to interact with its environment. For example, a method that writes into a log file is an exit point and the method that receives from the API is the entry points. There are different points of entry/exit like User Interfaces (UI) points, HTTP headers, and cookies, APIs, Files, and Databases.

There is a recursive relationship between Attack Surface Analysis and Application Threat Modeling: changes to the Attack Surface should trigger threat modeling. (Attack Surface Analysis Cheat Sheet | OWASP, no date)

Many tools could be involved in this process to identify the attack surface to crawl the application and figure out the exposed endpoints like OWASP ZAP[8], Skipfish[9], burp suite[10] and some development tools like debuggers may also help.

2.6.3 Threat modeling tools

Threat modeling methods is a security technique, it is used to create an abstraction architecture of the system to identify potential attacks that may arise. It helps in understanding the system and prioritizing the issues during the security process. (Shevchenko et al., 2018)

We have many threat modelings tools which depend on many methodologies to cover different type of systems.

- STRIDE

It as an acronym for six major categories of security threats. It evaluates the design of the system and categorizes the threats into the previously mentioned categories. (The STRIDE Threat Model | Microsoft Docs, no date)

It was firstly discovered in 1999 by Microsoft and adopted by them in 2002, it is implemented as part of the Microsoft Security Development Lifecycle (SDL). In this, data flow diagrams

[8] https://owasp.org/www-project-zap/
[9] https://tools.kali.org/web-applications/skipfish
[10] https://portswigger.net/burp

(DFDs) between the entities in the system are built. STRIDE is currently the most mature threat-modeling method. STRIDE threats and its related violation are shown in Table 2.

Table 2 - STRIDE threats and its related violation (*STRIDE chart - Microsoft Security*, no date)

Threat	**Violated property**
Spoofing	Authentication
Tampering	Integrity
Data repudiation	Non-repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

- DREAD

Microsoft also developed this threat assessment approach, which used to calculate, rate, compare, and prioritize the severity of risk for each threat that is classified using STRIDE. It calculates the risk using the following equation:

$$DREAD\ Risk = (Damage + Reproducability + Exploitability + Affected\ Users + Discoverability)/5$$

Each variable in the equation is a value between 0 and 10, and the final value between 0 and 10 as well. (*Threat Modeling | Microsoft Docs*, no date)

- Attack tree

It is one of the oldest and most traditional ways in threat modeling, in attack tree we represent attacks against the given system in a tree structure diagram, the root of the tree is the goal and the nodes represent sub-goals and there are different ways of achieving that goal with logical sequence.

The attack tree provides define the logic by using AND nodes to represent a logical sequence of steps to get to the goal and OR nodes to represent the alternatives. (Saini, Duan and Paruchuri, 2008).

- Trike

It is a framework for security auditing that was found in 2005, Trike Analyzes the relations between the actors, assets, intended actions, and rules. It starts with identifying them, then building a matrix represent the actors and assets and then evaluate the CRUD operation that is assigned to each actor in the system.

After building the matrix, an iterated operation through the DFD model should be conducted to assess the risk of attacks that may affect assets. (Saitta, Larcom and Eddington, 2005)

2.6.4 Static code analysis tools

It is the process of scanning the source code trying to find any potential security flaws; Code analysis tools search into the code that was developed in the development phase to identify defects, helps verify that developers are following guidance, and helps identify problems early in the development cycle.

Many commercial, free, and open-source tools are in the market that supports different programming languages. The following are some of them that support java and javascript, the main two languages that are used in Msg Life.

- Sonarqube

Is an open-source platform developed by SonarSource for continuous inspection of code quality to perform static code analysis to detect bugs, code smells, and security vulnerabilities.

It can be integrated with continuous integration tools like Jenkins and Atlassian Bamboo. Also, we can easily define a quality gated for implementation review and integrate that into the CI/CD pipeline as shown in Figure 7. *(Code Quality and Security | SonarQub>*, no date)

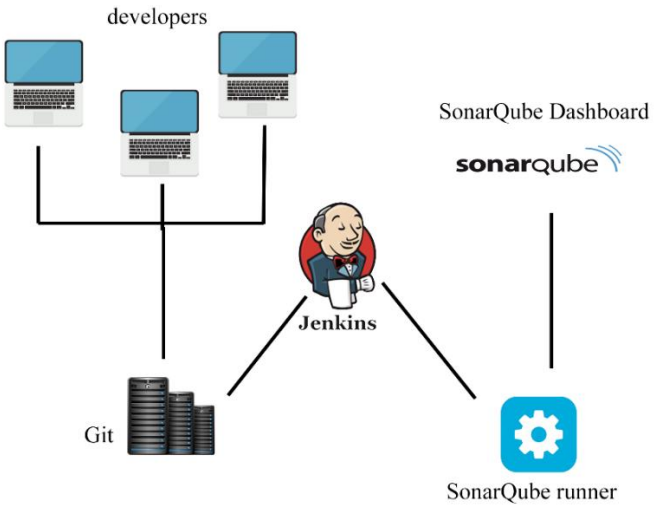

Figure 7 - SonarQube integration with CI

- Deep Dive

Static analysis tool for byte code to assess JVM-based deployment units (Ear, War, Jar, APK). It has many built-in features to search for code quality and security defects. (*Discotek.ca Deep Dive*, no date)

FindSecBug

This tool also works on identifying weaknesses and bugs in Java-based code. It can detect 135 different vulnerability types according to the pre-defined Bugs Patterns. It has plugins for maven, eclipse, IntelliJ, Netbeans, and Jenkins. *(Find Security Bug>*, no date)

2.6.5 Automated security testing tools

A wide number of Dynamic Application Security Testing (DAST) tools are available and all these tools have their strengths and weaknesses.

- OWASP ZAP

An open-source tool from OWASP, it provides web application security testers with many build-in features and extra plugins to perform fuzzing, scripting, spidering, and proxying to attack web applications. (*OWASP ZAP*, no date)

OWASP ZAP can be integrated with continuous integration pipeline to run an automated test against the application before release and there is a special Docker image that provides an easy way to run ZAP, especially in a CI/CD environment.

- Burp Suite

It is an advanced and one of the most well-known security tools, it is an integrated platform for performing security tests. It contains many features to support security actions from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities. *(Burp Suite - Cybersecurity Software from PortSwigge*r>, no date)

Burp has a free version, an enterprise, professional, and enterprise releases. Only the enterprise version can be integrated with the continuous integration pipeline and can work in headless mode.

- Arachni

It is an open-source framework to evaluate the security of modern web applications. It is a very high-performing, easy-to-use. It offers a web user interface that permits multiple administrators to manage scans.

It is very quick and generates a well-formed HTML report. However, the available scan options are so customizable that it would be great to integrate with continuous integration pipeline.

2.6.6 Dependencies checking tools

Open source is awesome and powerful but depending on so many open source libraries and components without keeping an eye on the security potential issues that may arise is dangerous. So, many tools were found.

- OWASP Dependency Check

According to OWASP the Dependency Check[11] library can detect publicly disclosed vulnerabilities within a project's dependencies.

It performs this by determining if there is a Common Platform Enumeration (CPE) identifier for the dependency. If found, it will generate a report linking to the associated CVE entries.

It also provided with plugins to integrate with CI pipeline and Maven (*OWASP Dependency-Check*, no date)

- OWASP Dependency Tracker

Dependency Track[12] is a Supply Chain Component Analysis platform that helps in identifying and reducing risk from the use of third-party and open source components. It implements a concept of Software Bill-of-Materials (BOM) which is a description of the components presented in the codebase with their licenses, version, and patch status. (Schütz and Salecker, 2008)

Two main formats for BOM exist today, CycloneDX[13] format, a specification designed for use in application security contexts, and supply chain component analysis. And the other one is SPDX[14] Which defined as a standard for communicating software bill of material information.

It can monitor components usage across all versions of every application. The platform can be integrated into Continuous Integration (CI) and Continuous Delivery (CD) environments and it supports many repositories like maven and NPM. *(OWASP Dependency-Trac*ker, no date)

Dependency Track offers easy integration with many vulnerability databases like NPM, VulDB, NVD, the National Vulnerability Database, which is the largest publicly available source of vulnerability intelligence. There are over 100,000 documented vulnerability in the NVD[15] from

[11] https://owasp.org/www-project-dependency-check
[12] https://dependencytrack.org/
[13] https://cyclonedx.org/
[14] https://spdx.dev/
[15] https://nvd.nist.gov/

the 1990s until now. Also, with Sonatype OSS Index vulnerability database[16], OSS Index does contain many vulnerabilities that are not present in the NVD.

2.7 Background and related work

This section discusses and Analyzes existing research and studies contributed to integrating security with agile methods to develop secure software products.

The secure software development literature field is very immense because many techniques have been suggested to get security activities along with agile methods (Khaim *et al.*, 2016). A bulk of earlier research in this field seems to be a comparison between the mostly known secure software development methods. like MS-SDL, Common criteria, and touchpoints in order to identify the potential integration with agile in (Singhal, 2014).

Many studies were conducted as well to measure the effect of security on agile. In this paper (Keramati and Mirian-Hosseinabadi, 2008) the agility degree of security activities is calculated based on simplicity, customer interaction, documentation, change tolerance, speed of execution, informality, people-orientation, iteration and flexibility. According to the study, building security teams, security education and guidance, and static code analysis are the top agilic security activities.

Another approach called FISA-XP was suggested by (Singhal, 2014), this approach considers two main measures in the selection process of the required security activities, the security activities agility level and its effectiveness. The research exposes a method to calculate a selection rate value based on those two factors. Hence, security activities with a higher selection rate should be implemented. However, aligning the cut-off line that separates the higher selection rate activities from the total set of activities depends mainly on the project and customer's needs.

This approach was examined on five activities from OWASP CLASP in XP model, concluding that Performing security analysis of requirements has a very high selection rate, specifying operational environment has a high selection rate, identifying user roles and requirements has a medium selection rate while detail misuse cases as well as identifying global security policy has a low selection rate.

Nevertheless, a few studies were published based on a real case study, (Dawson et al., 2010) made an early contribution. In their article, they reveal the risk of delegating security to the test phase and proposed a methodology for integrating software assurance into the Department of Defense Information Assurance Certification & Accreditation Process (DIACAP) development lifecycle that used waterfall development model and committed to The International Information Systems Security Certification Consortium, Inc (ISC)2 security best practices.

(Mougouei et al., 2013) discusses the risky influence of missing documentation in SCRUM development model on security considerations. The paper claims a new enhanced version of SCRUM called S-SCRUM, that aims to integrate security analysis and design activities into SCRUM process.

[16] https://ossindex.sonatype.org/

The supposed model focuses on web services. So, in order to validate it, it was tested on a money transfer service. However, other aspects on like other security considerations incorporate with SCRUM was not covered.

(Kainerstorfer, Sametinger and Wiesauer, 2011) in their study aim to experiment the effectiveness of MS-SDL for small teams, on a small product, in a real-life case study. Even though the case study was not fully complied with the MS-SDL, they successfully pushed the security to early stages in the development process and the target of developing a secure product was achieved. The study concluded that it is possible to incorporate security without significantly increase the development cost.

Another recent research in this field (Nunez, Lindo and Rodriguez, 2020), the paper proposes a new flexible model in secure software development for agile, called Viewnext-UEx, this model was examined in Viewnext corporation. During the study, six secure development models were investigated including OWASP CLASP, MS-SDL, BSIMM, SAMM project to find the intersections between them. They build their model upon these cross activities. As a result, the model is structured in a way similar to BSIMM and SAMM project. It consists of four main development areas (Policies, secure development methodology, supervision, and observation). By applying the model, they could successfully decrease the time spent on fixing vulnerabilities and the number of vulnerabilities in the mentioned organization by 68%.

3 Value Analysis

Building things is easy but building the right thing that fits the market, the customer, and the community the most, is the hard thing to achieve. In this chapter, we will highlight some of the topics that are related to the value of this study.

3.1 Innovation / New Concept Development (NCD)

In this section we will use the NCD model which was proposed by Koen et al. in 2001, to structure our front-end ideas in security methodologies, and end up with a well-formed mature security assurance process.

The NCD is a terminology to understand the front-end of innovation it entails three main parts, five core activities “Front-End of Innovation (FEI)”, the engine that powers the elements, in the middle of the model, and the external elements that influence the innovation process “Influencing factors”. As shown in Figure 8 (Koe*n et al*, 2001)

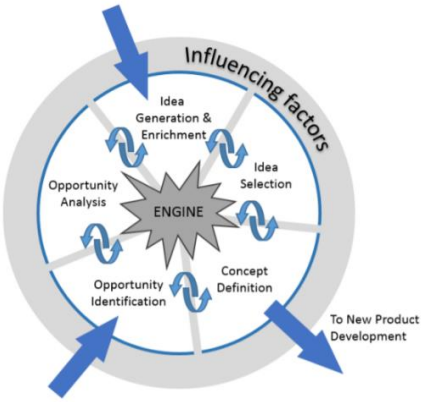

Figure 8 - The NCD model (Koen et al., 2001)

3.1.1 Opportunity Identification

In their article, Koen et al. define the opportunity identification by recognizing the new opportunity the company may face by design or default. (Koe*n et al*, 2001)

And according to that definition, we can clearly acknowledge that software security has been always a hot topic and the objective of software companies that straggle toward making their products immune to the attackers. The main idea of this thesis comes from Msg Life Company that wants to improve the security of its main product after they discover some issues and flaws in their core software which is the cornerstone for many insurance companies around the world. They notice that they don't dedicate actual stories or tasks to ensure security, while they focus on the functionalities of the product that they need to deliver to the customers and they concluded the importance of integrating security as a part of their agile development lifecycle. That is reasonable because agile does not enforce security by default and they need a dedicated insurance security plan to perform that.

The same situation applies to 97% of the companies worldwide that adopt agile partially or fully in 2019, this agile companies face a different level of security risks depending on many internal and external factors. Disregarding the verity of this wide vector of differences, the first and most effective defense line comes from inside the company to develop the most secure software as possible and this can't be done in ad-hoc basis, it requires some roadmap to obtain the most from agile as well as security practices. (*2019 Survey on Agility Agi l e Transformati on From Agile experiments to operating model transformation: How do you compare to others?*, 2019)

3.1.2 Opportunity Analysis

Opportunity analysis, where we discuss in more detail the trends of the opportunity and try to shape it into a business idea to give the company an advantage over its competitors. (Koen et al., 2001)

Introducing security to the company comes into many forms but putting security in a robust methodology can help the company grow with less risk. There is a variety of security solutions that prove its abilities to improve security awareness and show its efficiency in introducing security to the company. This security assurance plans are proposed by the security community depending on the company's size, architecture, products, existing SDLC, objectives, and budget. As we discussed in the existing SSDLC models previously in chapter 2, some of this well-known security assurance methods that may be applicable to Msg Life company. In Table 3 we compare them into more concerning details that may affect the selection of the solution. (Mohammad, Alqatawna and Abushariah, 2017)

Table 3 - comparing between SSDLC models

	MS SDL	**Touchpoint**	**OWASP CLASP**	**SAMM project**
The nature of the model	Heavyweight	Lightweight	Lightweight	It depends
Activities number	16 activity with optional activities	7 activities with 1 optional	24 activity	12 activity
Organization size	Large	Small to large	Small to large	Small to large
Activities dependency	Dependent	Independent	Independent	Independent
Easiness of adaption	Difficult	Easy	Easy	Easy
Education and training	Dedicated activity	None	Dedicated activity	Dedicated activity
Documentation and guidance	Rich resources	Weak resources	Rich resources	Rich resources
Security testing	White box	Black box and white box	Black box	White box and black box

3.1.3 Idea Generation & enrichment

This process represents the idea creation, evolution, and growth over time. When the opportunity starts to evolve into a clear coherent idea, we can receive a business benefit from it. (Koen et al., 2001)

The original idea beyond this thesis was simple. It was making the core product at the company more secure.

From the company's perspective, doing a penetration testing or in the best scenario, performing it with implementing some source code analyzer like SonarQube was the most obvious approach, that was going to fulfill their objectives. But, after assessing of the situation in the company, diving deeper, searching for the root of the problem and checking the security maturity level, new problems were exposed, problems like unawareness of security best practices from the developers and stack holders, the rapid changes in the system which

enforce changes in the attacker surface -thanks to agile model-. De-emphasizing on documentation activates. Under these circumstances, it became obvious that penetration testing is not useful as a late threats' prevention defense mechanism and its result will be outdated soon according to existing the development model.

Due to those reasons, a solution that reflects a consistent gradual change in the company was needed, and we started to step forward securing the development lifecycle via some security assurance method.

3.1.4 Idea Selection

Selecting the idea or approach between the set of choices that return the maximum business value is considered very critical and influential. (Koen et al., 2001)

After discussing the situation with the stakeholders, analyzing the opportunity and comparing between the well-known existing methodologies and tools in the secure development field, we found that Microsoft SDL was heavyweight to implement especially when it is most appropriate to large products and large teams; it was built around the way they are building Microsoft Windows in big releases. This is not the case in Msg Life company. Furthermore, it does not take you from you are to where you want to be. So, it is not a roadmap. Also, it misses a few points, because it does not have a specific activity for operation and governance and it lacks effective activities in the implementation phase, mainly because it focuses more on security as a supporting quality.

While, Touchpoint is a high-level map and goals, without enough details to execute and it doesn't define what we should do to achieve that goal, which works fine with high-security experience experts, Thus, as Microsoft SDL, Touchpoint doesn’t seem so effective in Msg Life, where there is no dedicated security team.

OWASP CLASP in its turn, is considered a large collection of activities without priority order, and it is also good for experts to use as a guide.

The final option, SAMM project that equipped with rich documentation, security assessment tools, and improvement metrics. SAMM project looks like the optimal approach to follow that enables the company to select what they want to improve in a timely controlled roadmap.

And the decision in the company was made to follow SAMM project v1.5 methodology that fits the objectives and benefits from the DevOps tools that already exist to integrate some security behaviors with it, so less impact on the agile flexibility is required. Due to its continuous nature and not a thing, you do once in a lifetime.

3.1.5 Concept definition

This process represents creating and developing a business case. (Koen et al., 2001)

Following SAMM project security assurance plan and integrate it with existing SDLC that already exists in the company, will cover the objectives of Msg Life in improving security level in its products, and more, it will be able to ensure security is being respected in its future products. And security will not be a barrier, instead, it will part of the day to day tasks.

3.2 Solution's Value

3.2.1 Value

Analyzing the value of the product or solution is an approach to ensure that our idea is optimal in comparison with other alternatives to meet the customer's needs. (Tian *et al.*, 2007)

The value from this work comes from the safety and mitigating security risks that face Msg Life organization from the early steps in the development lifecycle, protecting its confidential information, also enabling application safe operation, which by result, saves the company's assets, software, business, and repudiation.

3.2.2 Customer Value and Perceived Value

The concept of perceived value represents the customer's notion of the value of an offer. (Heinonen, 2004)

In other words, the value is a balance between the benefits and sacrifices that a given offer or idea has from the customer perspective. And whether the idea is worthy or not. And this is in fact the same principle in security, risk, or sacrifices acceptance, and whether we accept this risk level, and depending on that a set of security restrictions is required.
Customer value in this context is the perception of the required security level in the development environment and the lifecycle in terms of time and effort and financial requirements.

Taking into consideration that the company has not yet been under attack especially when its products for internal uses in in their customers' environments which may seem safe and less dangerous.

This may seem unconcerned at the first sight, and security practices are just an overwhelming for agile. But if we investigate further, and look at the statistics globally, we find out that 42% of the small companies suffered from attacks in 2019. And 34% of attacks are insider ones which are more harmful and can cost a company up to $8.76 million a year. (2018 Cost of Insider Threats: Global Sponsored by ObserveIT, 2018).

These statistics give us more obvious vision of the actual situation that for sure seems concerning. And we can conclude the value of this project where we avoid the company risks

that might cost the company money and repudiation and in the worst-case scenarios; fines under GDPR regulation.

3.2.3 Longitudinal Perspective of Value

The longitudinal Perspective of Value that presented by Woodell which identifies four distinct positions for the customer value, as shown in figure 9 and the pros and cons for each of them (Woodall, 2003)

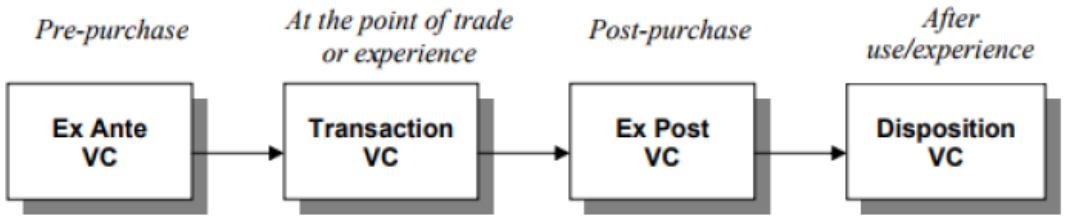

Figure 9 - Longitudinal perspective on VC (Woodall 2003)

Table 4 presents the benefits and sacrifices from the pre-purchase phase to after use/experience phase for Msg Life.

Table 4 - Benefits and sacrifices of the customer along the process

Stage	Sacrifices	Benefits
Pre-purchase	• Time. • Effort.	• Adapting to different contexts. • Increase knowledge.
Point of experience	• Time: Slower sprints in agile due to extra security load. • Effort: need to update or establish a bunch of documentations, in a lot of security-related fields at the company level like developing a business profile with the managers. • Cost of a developer's security training. • The cost and time for upgrading some legacy systems and tools.	• Introducing security best practices to the whole company. • Securing the undergoing products. • Establishing security behaviors to build above it (threat modeling, Analyzing attack surface, automated security tests, static code analysis... etc) for the core products of the company. • Establish a security-awareness mentality for developers.
Post-purchase	• Effort and time to maintain the established practices to fit new software.	• Securing future products. • Pre-defined security roadmap for the next year in the company.

		• Mitigate common vulnerabilities in the source code and the final product. • Security becomes a behavior, not a load.
After experience	• The time and effort to maintain the security practices in place. • The cost of annual security training for developers that might be expensive if the company decided to move one more level and conduct special training for each developer based on his position.	• Low level of vulnerabilities. • Less attack surfaces. • Minimum impact in agile speed. • Enhance the overall security maturity level in the company. • Security becomes part of the development and developers' culture. • Raise the readiness in worst-case security incidents scenarios • Minimal impact worst-case scenarios. • Products are secure by default. • Encourage investors.

3.3 Value Proposition

A lot of definitions for this term are widely used since it was first mentioned in 1988, but all of them refer to the same thing.

According to (Tuominen, 2004), “competitive advantage and superior value flow from whatever unique ability a firm has to shape, reshape, configure and reconfigure those assets to serve customer needs”.

The value that this thesis can be summarized into putting a roadmap that the company can follow for the one year that can ensure security in the company’s products. This roadmap requires establishing security behaviors in the company based on the latest technologies and methodologies in this field. These practices and behaviors can easily integrate with agile and SCRUM software development.

3.4 Quality Function Deployment (QFD)

QFD is a methodology it was first incepted in Japan in the early 1970s, it is a measure of customer satisfaction with a product or a service, used to ensure matching the customer’s needs into the design and the effectively responding to those needs and expectations. (Akao and Mazur, 2003)

For this work, the House of Quality matrix was developed, the widely used form of QFD that consists of six components, the customer requirements, technical requirements, a planning matrix, an interrelationship matrix, a technical correlation matrix, and a technical priorities/benchmarks and targets section. (Tapke *et al.*, no date)

To define the target values in the HQD that is presented in Figure 10, we used the maturity values that were collected from the conducted security maturity evaluation test for Msg Life, where we used SAMM project security assessment tool.

The result of this assessment was a value in each of the 12 practices of SAMM project, this value represents the maturity level in this practice. Based on this maturity level, the target maturity level in these practices was defined. And the full description for the methodology and selected targets will be found in chapter 4.

The comparison in the technical evaluation section of the HQD was developed between Msg Life and the average maturity values for Software Companies in general and to the companies that provide services to the insurance sector that is like Msg Life.

This data was collected from BSIMM 10 survey which is a study of real-world software security initiatives, the model was built out of data observed in 122 firms, and 11 firms of them are insurance companies. (Migues, Steven and Ware, 2019)

3.5 Conclusion

Careful examination and deep analysis of the proposed methodologies derives to select SAMM project framework. This project is going to be applied in Msg Life organization since SAMM project fits most of the claims.

Based on that model, a security assurance roadmap will be formed. This roadmap will specify the required set of activities to improve the security maturity level in the organization. Furthermore, incorporate these activities with the current lifecycle, making the current and the future products more secure.

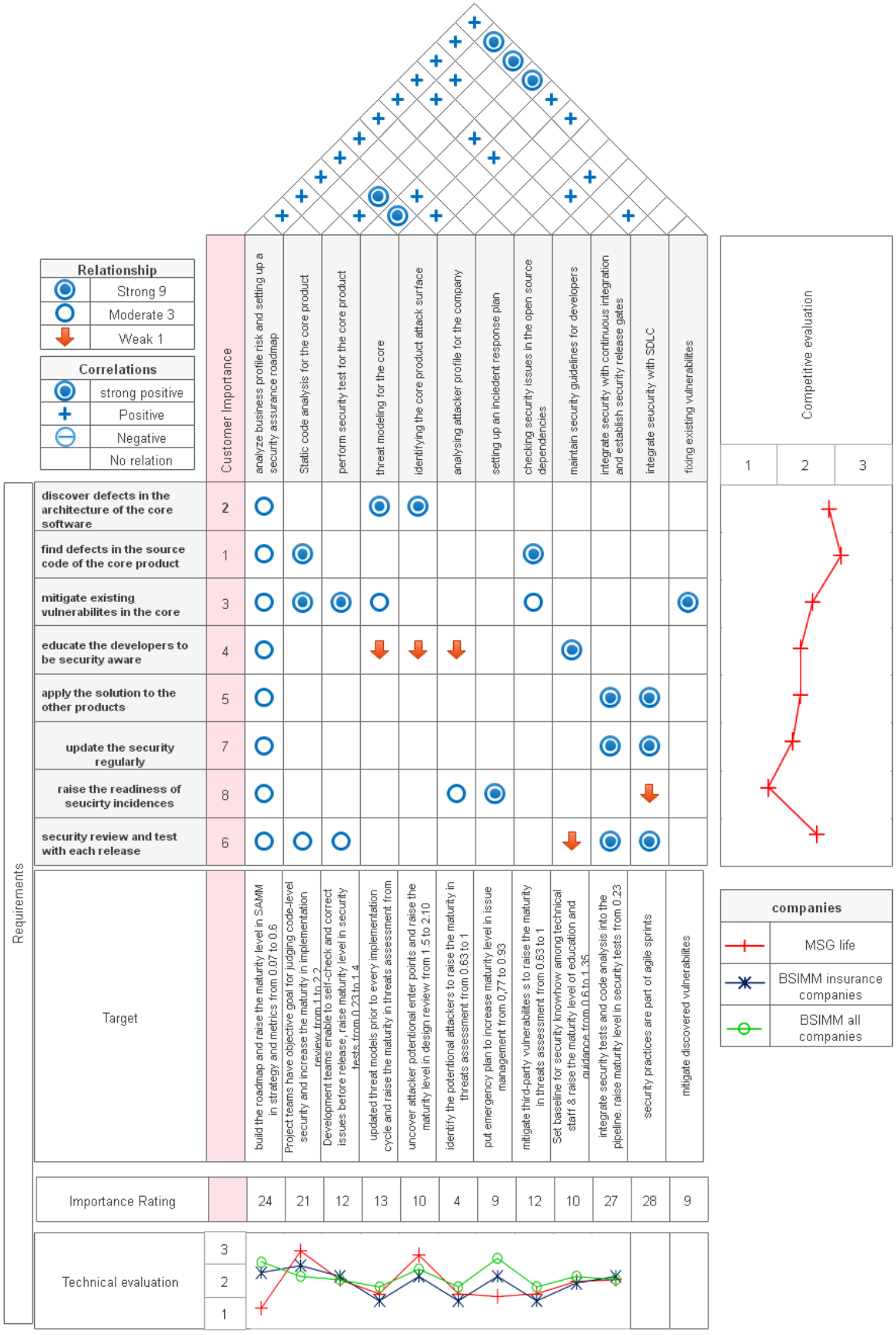

Figure 10 - The house of quality for Msg Life security assurance

4 Design the SAMM project solution for Msg Life

In this chapter, the detailed design is presented based on SAMM project approach. Starting with explaining the iterative model that we used, the stages in each iteration, and the criteria that are used to select the set of activities throughout the roadmap.

4.1 Security assurance roadmap applied model

The roadmap timeline was established based on SAMM project approach as shown in Figure 11. Starting with the preparation phase to define our scope, which is the core platform in the company and all customers' projects that were built upon it and identifying is our main stakeholders based on this target scope.

Then perform the assessment to verify which security activities already exist in Msg life, this assessment is conducted by interviewing stakeholders in the organization and audited to verify the correctness in each of the 12-security practices. In Appendix B the full list of the assessment survey questions is attached. The result of this assessment is a numerical value in each security practice based on SAMM project scoring model where the highest score is 3.

It is followed by defining our target level of maturity. Eventually, a roadmap entails the selection of which practices to improve in each planned phase is set in place.

After establishing the roadmap. Collaborative work should exert to achieve the stated levels by performing the prescribed activities.

At the end of the first phase, the roadmap could be adjusted based on what was accomplished and what is emerged during the first phase, to start the next phases.

In our solution, the roadmap is divided into four phases each of them is three months long. the first two phases are thesis oriented they will be described into details in the following section while the other two are optional to the company to commit to. Figure 11 shows this timeline and roadmap's four phases.

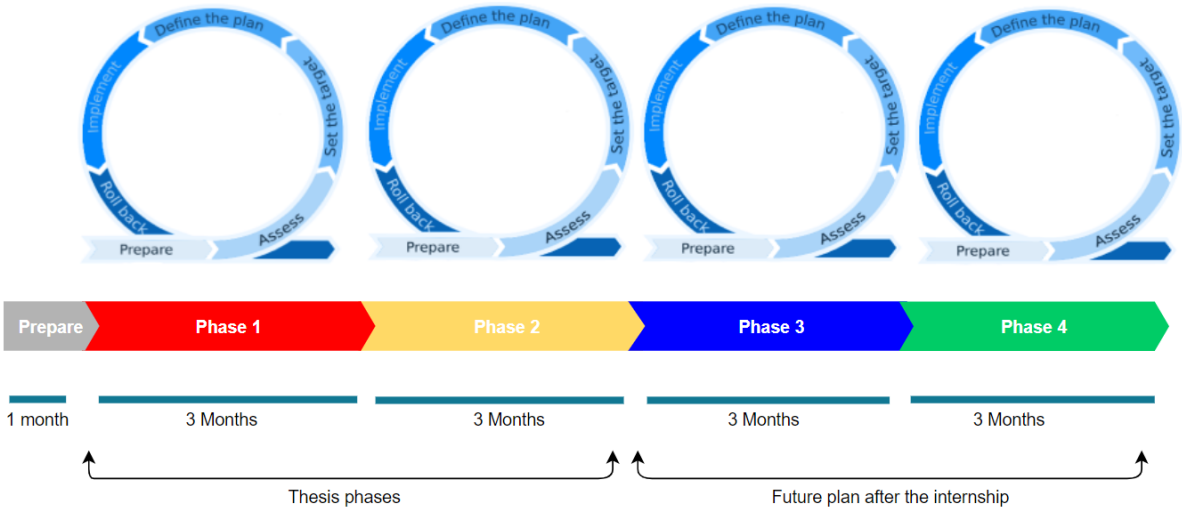

Figure 11 - The security assurance roadmap phases in a timeline

Many factors should be considered to identify a realistic maturity level. The main reasons can be summarized in the following points:

a) The size of the organization, Msg Life Iberia team in Portugal is a small team, there is no dedicated security team nor DevOps team. Which implies two main restrictions, the first one is the limited guidance in security inside the company. And establishing continuous security relies normally on other fields especially the DevOps field. So, starting from this bounded situation affects the speed of security progress.
b) The type of products. Msg Life main products are insurance software applications with a complicated architecture, this software works internally, in most cases, in their costumer environment which is considered out of Msg Life scope. This means improving security in the production environment is not among the critical things that we need to worry about. Especially during the first and the second phase of the roadmap.
c) SAMM project recommendations, the project comes with a "how-to" guide. It discusses some actual examples of organizations, and some recommendations for similar issues and targets that should be considered while building the roadmap.
d) The current situation of the company, which represented by the initial maturity level, with attention to the priorities and dependencies in the model.
e) Development approach in the company, technologies that are already in use and budget, these factors also impact the decision for a certain level.
f) Msg life needs, perspective, and priorities. Some targets are more coherent with the organization's business roadmap. To achieve that the roadmap was discussed with the organization's main stakeholders to make sure that it fits their perspective and meets their goals.

g) Some security practices show a high level of agility and automation, meaning that they can get along with agile easily, while the others overwhelm agile process. Thus, the decision of the needed security activities affected by their level of agility, which was the topic of many papers in this field.
An example of that is displayed in table 5, where the requirement column represents 22 security assurance requirements associated with their security level that goes from basic (1), heightened (2) to high (3), and the automation level for these requirements. And finally, the level of adaption in three main agile methodologies which are extreme programming, Kanban, and scrum, the one that is used in our case. (Rindell, Hyrynsalmi and Leppänen, 2015)
Even if the table discusses the automation level for MS-SDL. But it indicates the potential practices to be implemented in our solution and the way they could affect the agile process.

Table 5 – Agile methodologies compliance with security requirements (Rindell, Hyrynsalmi and Leppänen, 2015)

No.	Requirement	Lvl.	Aut.	XP	Scr.	Kan.
1	Application Risk Analysis	1	M	I	I	I
2	Test Plan Review	1	M	A	A	A
3	Threat Modeling	1	S	I	I	I
4	Goal and Criticality Definition	2	M	A	A	A
5	Business Impact Analysis	2	M	A	A	A
6	Documentation of Security Solutions	2	M	A	A	A
7	App. Security Requirement Definition	2	M	I	I	I
8	App. Security Settings Definition	2	M	A	A	A
9	Security Testing	2	A	I	I	I
10	Security Auditing	2	M	A	A	A
11	Arch. and App. Devel. Guidelines	3	M	A	A	A
12	External Interfaces Review	3	M	I	I	I
13	Use of Secure Design Patterns	3	M	I	I	I
14	Attack Surface Reduction	3	M	I	I	I
15	Architectural Security Requirements	3	M	I	I	I
16	Internal Communication Security	3	S	I	I	I
17	Security Test Cases Review	3	M	A	A	A
18	Test Phase Code Review	3	M	I	I	I
19	Use of Automated Testing Tools	3	A	I	I	I
20	Security Mechanism Review	3	M	N	A	A
21	Development-time Auditing	3	M	N	N	N
22	Security training for Developers	3	M	A	A	A

M = Manual, S = Semi-Automated, A = Automated; I = Integral, A = Adaptable, N = Incompatible

h) To make the target maturity levels more reasonable, we checked on the maturity level in the real-world situation. So, the Building Security In Maturity Model (BSIMM) V10 was strongly considered. Figure 12 shows the average level of maturity in different types of software companies.

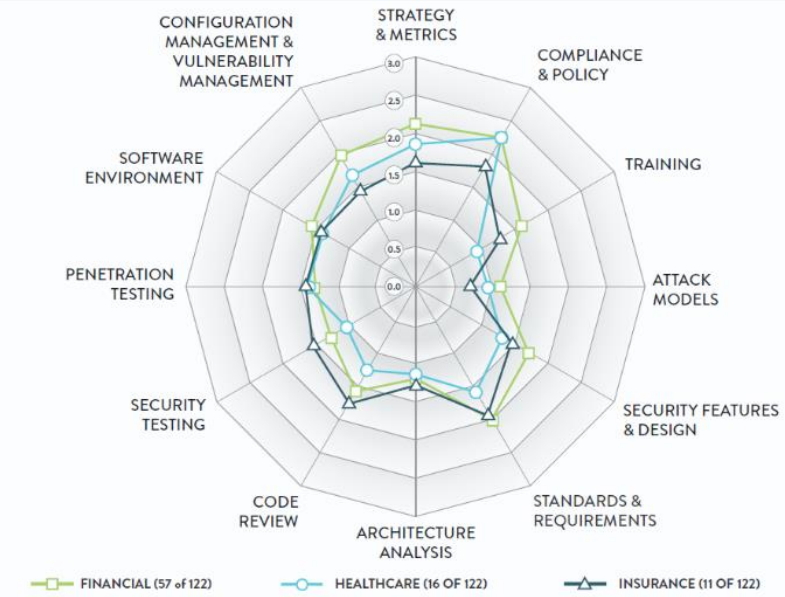

Figure 12 - BSIMM V10 Financial vs. Healthcare vs. Insurance Spider Chart(Migues, Steven and Ware, 2019)

4.1.1 Use case diagram

The security assurance road map use case diagram is presented in Figure 13 to clarify the actors, where the security team, managers, and business owners are our main actors. The security team, represented by the author, responsibilities starts from the initial phase and design the solution, the implementation part until evaluating the results. While managers, business owners, team leaders participate in the management part and partially in the implementation use case. Developers and architects are seen in implementing the solution.

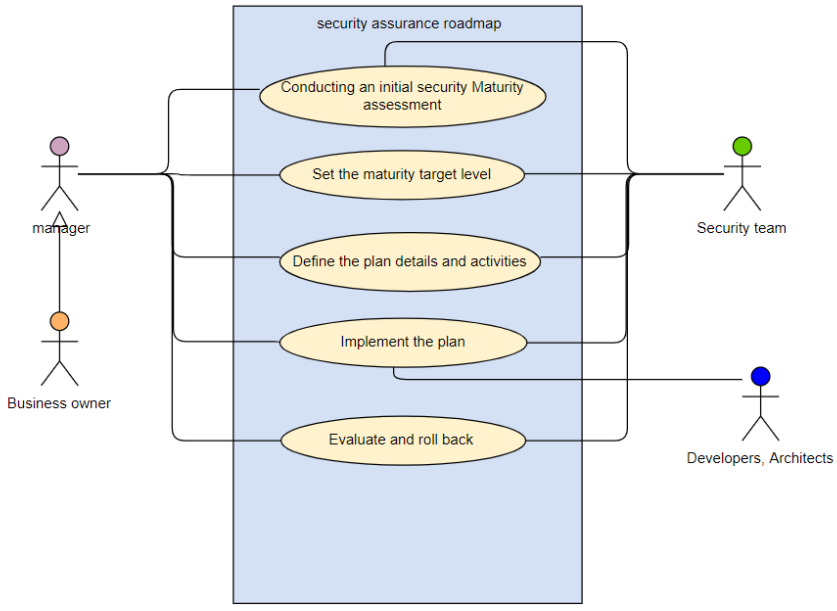

Figure 13 – Security assurance roadmap use case diagram

4.1.2 Activity diagram

The activity diagram of the security assurance roadmap is displayed in Figure 14, this flowchart shows activities performed to set the roadmap and improve it over time.

In the flowchart, the black circle symbolizes the beginning of the activity diagram. While the rectangles are the main building blocks, they Indicate the activities that make up the overall process. (*UML Activity Diagrams - Graphical Notation Reference*, no date)

The arrows between activities outline the directional flow, or control flow, of the activity. An incoming arrow starts an activity; once the activity is terminated, the flow proceeds with the outgoing arrow. (*UML Activity Diagrams - Graphical Notation Reference*, no date)

The decision node is represented by the diamond symbol, it represents the flow of control in branches based on the defined condition. The flowchart is terminated by the circle that serves as the final state. (*UML Activity Diagrams - Graphical Notation Reference*, no date)

Going back to Figure 14 flowchart, where the performed activities in the roadmap are outlined. Starting from initiating the security maturity assessment and auditing for the correctness of the assessment.

If the roadmap is already in place, meaning this is not the first phase, maybe some adjustments should be made to improve the roadmap before execution. Otherwise, much more effort should be made to establish a new one.

After defining the targets, setting up its corresponding security operations that lead to reaching the defined targets, the most time-consuming activity comes into place to put these security operations into actions.

The last activity in each phase is evaluating the results. This process is planned in our solution for four phases and implement two of them, but it could be iterated for as many phases as needed after that.

4.2 Conclusion

In this chapter, the design of the suggested approach to be followed in Msg Life is clarified. This design aims to maintain consistent security practices into a flexible security assurance roadmap.

This roadmap timeline cracked into four main phases with full consideration of SAMM project framework.

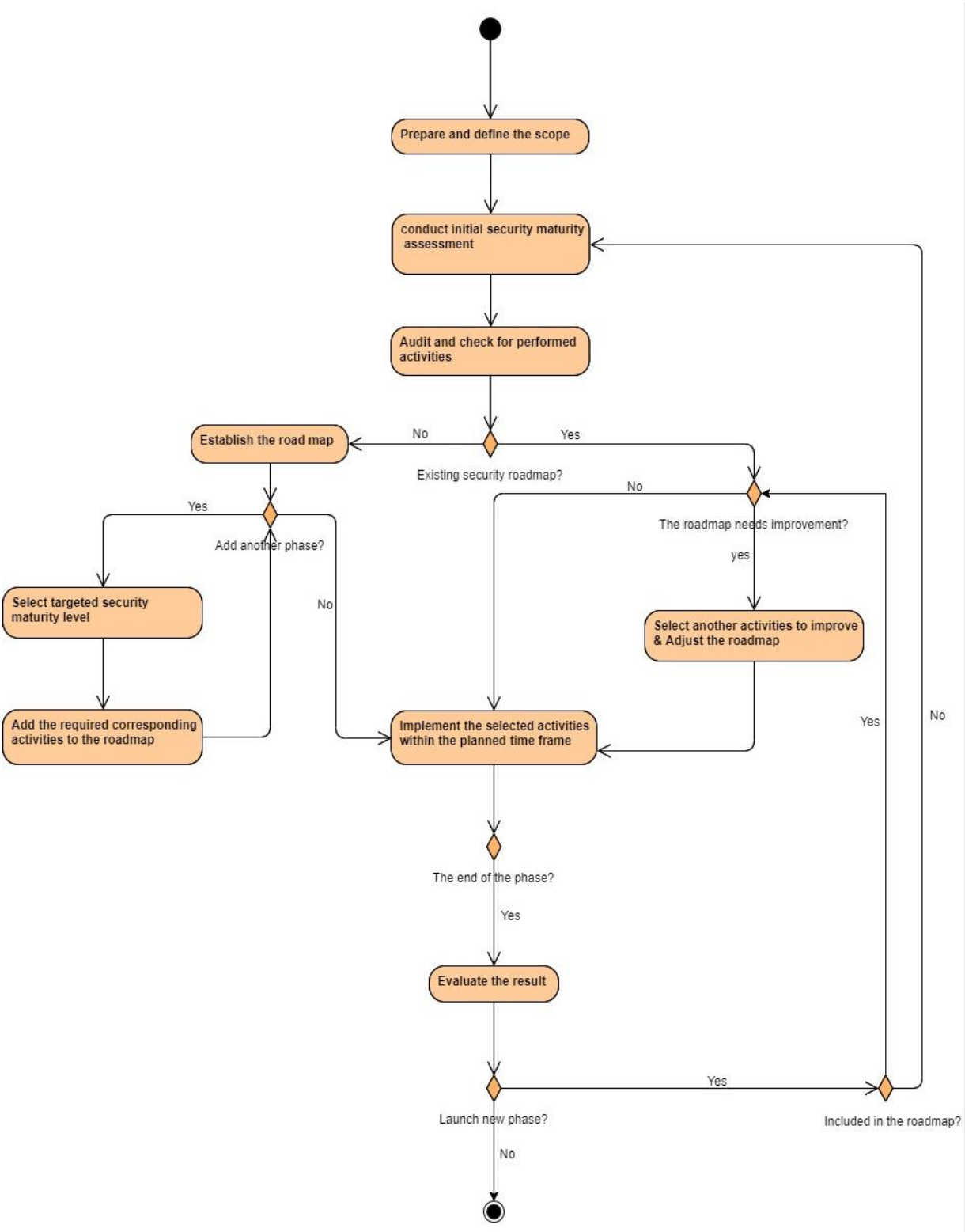

Figure 14 - Security assurance roadmap activity diagram

5 Implementing the solution

In this chapter, the roadmap represented by its four phases is explained after the initial security assessment.

Also, the target maturity levels and the detailed set of activities to be implemented during all the phases were stated as part of the first phase. The implementation of the first and second phases is also covered in this chapter.

5.1 Phase one – preparation

In the preparation phase, the scope of the roadmap is defined by the software artifacts which are to be included.

This group of artifacts consists of the core product in the company, in addition to all the customers' projects currently under development, that depend on this core product. The number of those projects is estimated by five.

5.2 Phase one - Initial security maturity assessment

SAMM project comes with a maturity assessment tool to evaluate the current security maturity of the target organization. The tool consists of questions that are used to evaluate the security posture of the organization. The questionnaire was sent to product managers, system architects, and developers in the company to assess the organization's maturity in each of the twelve security practices covered by the SAMM project.

Figure 15 shows the results of the initial maturity evaluation assessment under SAMM project four main security functions. The blue section in the chart, represents the governance function, the orange section is the construction function, the green is the verification one, while the red symbolizes the operations function.

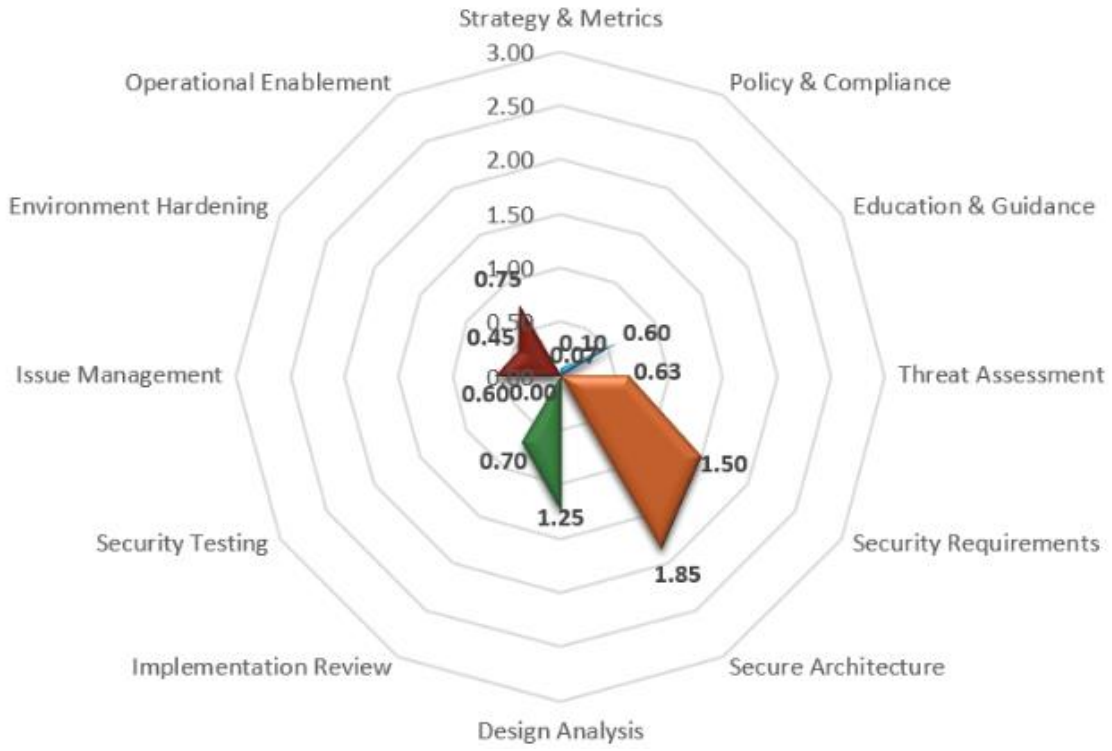

Figure 15 – Initial security maturity evaluation test results in Msg Life Company

As shown in the figure above, the company's level in **strategy and metrics** is 0.07 out of 3 due to the fact that the business risk profile is only considered partially with no available documentation of it.

Similarly, neither a security roadmap to plan the future security improvements nor risk categories to classify the application under development were in place. Moreover, the security expenses of the organization such as training fees, tools licensing, and security outsourcing are not tracked.

Similar results were seen in **policy and compliance,** where the company has not implemented any of the measures advised by SAMM framework except for allowing team leaders to request an external audit. These features, however, are not well utilized given that the company is not compliant with any known information security framework such as e.g. PCI-DSS. This led to the organization scoring only 0.10.

In **security and guidance**, the company scored 0.60, for implementing primitive security guidance across the company. A limited number of the development teams, however, understand the basics of secure coding and secure development best practices. While stakeholders can pull security coaches for consultation if needed.

Under the construction main function in SAMM project model, the company is doing some **threat assessment** through an informal and undocumented abuse-case analysis with minimal

awareness of threats and its related impact. However, high attention is paid to threats stemmed from external software and third-party dependencies. As a result, the company scored 0.63 in this metric.

On the other hand, the company is doing good when it comes to **security requirements** scoring 1.50 maturity value. This is because requirements are specified during development. Also, a matrix of roles and capabilities is documented and reviewed before the release of every project.

The company achieved the highest maturity value in **securing the architecture** where it scored 1.85. They already have a static list of recommended third-party software. Also, in this practice, architects and senior developers identify applicable and appropriate patterns for each project during the design phase, alongside the reusable code components based on established design patterns, in addition to shared security services that can be used within projects across the organization.

In the implementation function, there is a moderate **design analysis** behavior, where project-based teams specifically Analyze design elements. This analysis includes only the minimum categories, namely:

- Authentication
- authorization
- input validation
- output encoding
- error handling
- logging
- cryptography
- session management

The teams also define how the developed system should handle high-risk functionality (such as CRUD of sensitive data). However, this analysis is done, similarly to other security practices, in an unstructured and non-consistent manner, where a formal process is neither defined nor documented.

In the **implementation review,** a static code review analysis takes place in the Msg Life company using SonarQube. However, the static code analysis is not fully integrated into the organization's CI/CD pipeline leading to a situation where many teams chose to avoid using it to scan their code. This leads to the organization scoring 0.70 in this section.

In the operation function, the score of the **issue management** practice was 0.60, due to a minimalistic incident response plan, where the stakeholders know only the point of contact, but no formal plan is in place.

In the **environment hardening**, the organization fulfills some security requirements to the end customer who is responsible for the production environment. The organization also

recommends additional tools to the customer to protect the software while running in a live environment. The company scored in this practice 0.45.

Since a list of security features is built into the software, options for configuration, security impact, important security-related alerts, and error conditions are delivered to the end customer and there is no change management plan, the company scored 0.75 in **operational enablement.**

5.3 Phase one - Setting the targets

Starting with analyzing the result of the initial security assessment, and the effect of the previously mentioned factors in section 4.1, many meetings to discuss the roadmap were organized, and eventually cultivated to the targeted values shown below in Figure 16.

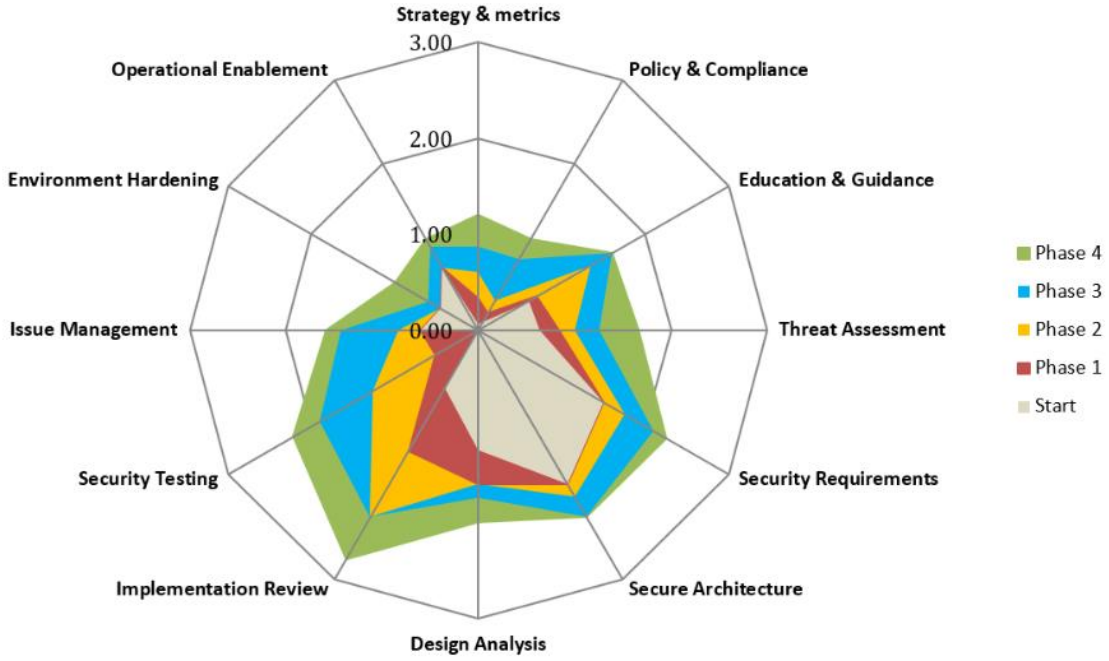

Figure 16 - Spider chart for the Msg Life roadmap in four phases

This stage was performed concurrently with the next stage. Starting from an overall maturity level of 0,69 out of 3 concluded from the initial assessment, the value of the maturity level metric after phase one is expected to increase by approximately 0.19, scoring 0.87 out of 3. After phase two, it is expected to reach 1.13. During the third phase, the maturity is expected to rise to 1.36 and finally to 1.63 after finishing the fourth phase. The full table of the exact targeted maturity values enclosed in the appendix A Table A 1.
Furthermore, the details of the plan are presented under section 5.4.

5.4 Phase one - Defining the plan

The full explanation of the required activities under each security practice is described in this section. This set of activities need to be implemented to achieve the target maturity level.

5.4.1 strategy and metrics

The outcome of every small change in this security practice would reflect on all divisions across the organization.

The following chart in Figure 17 represents the progress in this practice for the four phases.

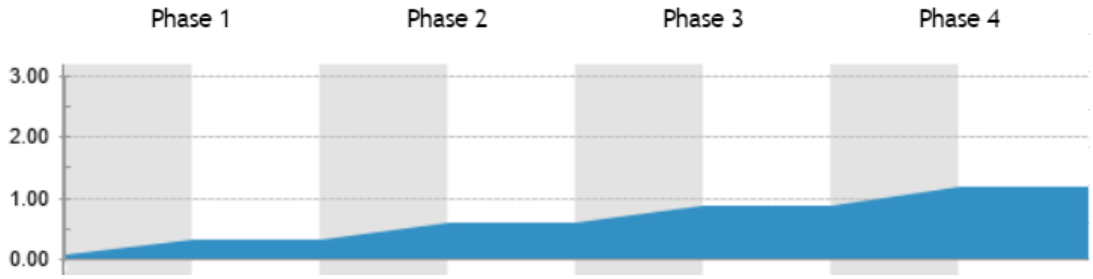

Figure 17 - progress in strategy & metrics in four phases

The main focus during the first and the second phases, as shown in Figure 18 and Figure 19, is to set a general security roadmap and define the company business profile risk, based on a list of worst-case scenarios that could lead to severe damage to the company's business portfolio. The goal is to analyze those scenarios from a security perspective, to evaluate their severity to the organization's business.

The analysis is done based on how critical each product is to the organization and the potential impact it may have on the organization if its security were compromised.

These threats and abuse cases have different levels of financial, legal, and reputational impact on the organization, where some could lead to violating regulatory constraints/restrictions such as GDPR regulation. Examples of those abuse cases include data theft, data tampering, and account hijacking.

Many tools can be used to perform a threat evaluation, such as OWASP risk rating framework, OCTAVE, or Trike method, as described in section 2.6.1.

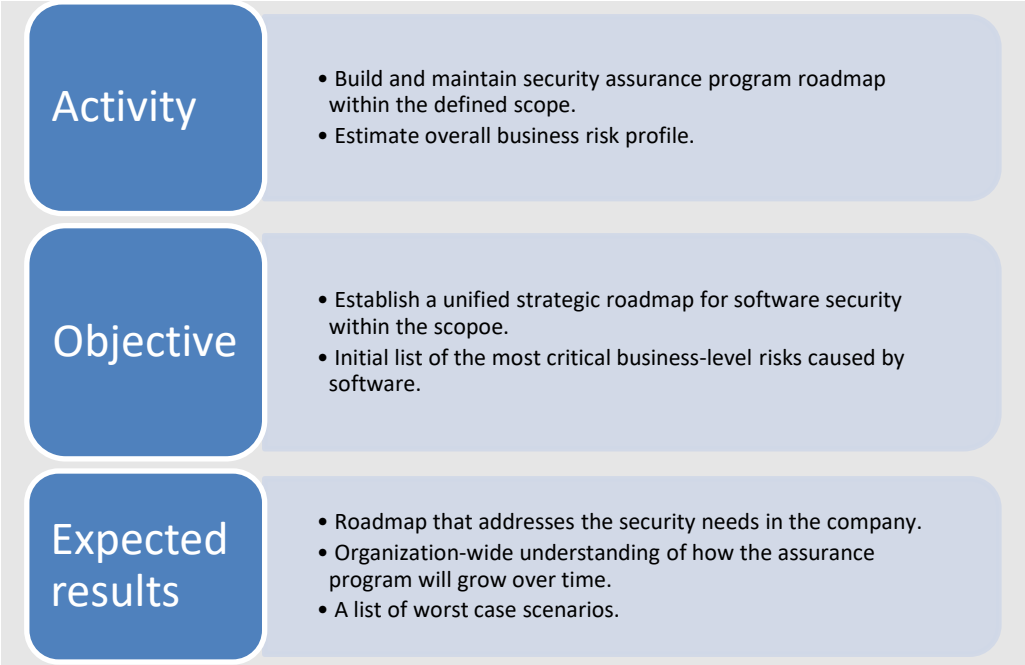

Figure 18 - strategy & metrics activities, objective and expected results in phase one

In the second phase, in addition to maintaining the security roadmap, work must be done to increase the awareness within the organization.

Additionally, categorizing a small set of software artifacts, based on their criticality, into (Low, Medium, High) groups is supposed to be done.

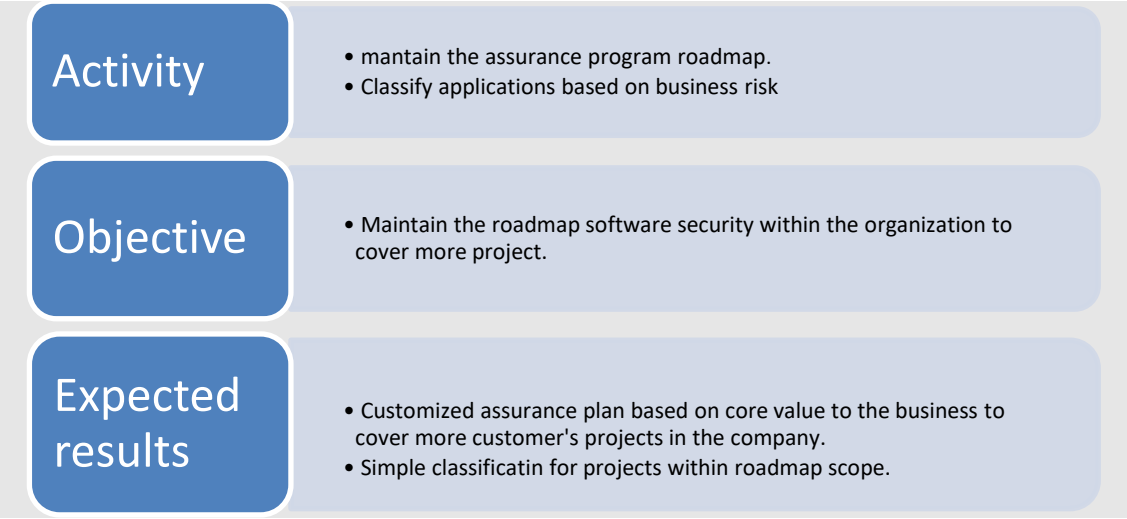

Figure 19 - strategy & metrics activities, objective and expected results in phase two

During phase three, the general target would be maintaining the same previously mentioned activities. But the main focus would be to adjust the assurance plan to cover at least half of the projects within the scope and map those projects into their respective risk categories.

By the end of phase four, most customers' projects should be covered in the roadmap and have their risks evaluated.

5.4.2 Policy and compliance

There are many standards and regulations which are used to assist and manage software development and operations environment in order to handle critical data, such as the Payment Card Industry (PCI) and Healthcare Information Portability and Accountability Act (HIPAA).

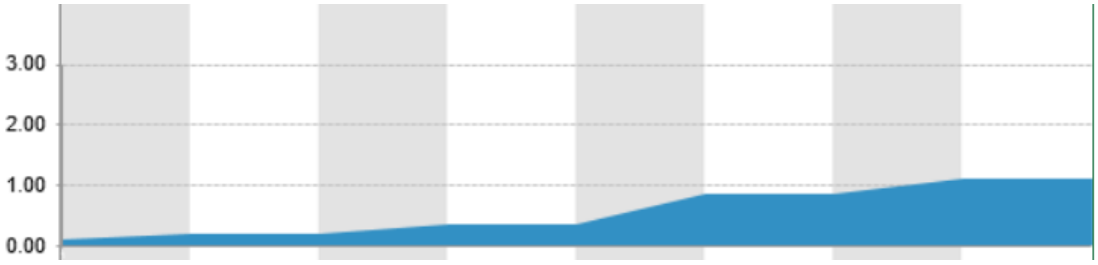

Figure 20 - Progress in policy and compliance within four phases

In our case, we are interested in HIPAA regulation, since Msg Life company provides its services to insurance companies which are covered by this regulation.

Becoming HIPPA complaint is not on the organization's short team goals, trivial progress can be accomplished in this practice, by making the main stakeholders aware of the current compliance driver during the phase one, and conducting unofficial assessment during the second phase of the road map, to give the stakeholders an overview on the actual compliance status within the organization. As shown in Figure 21.

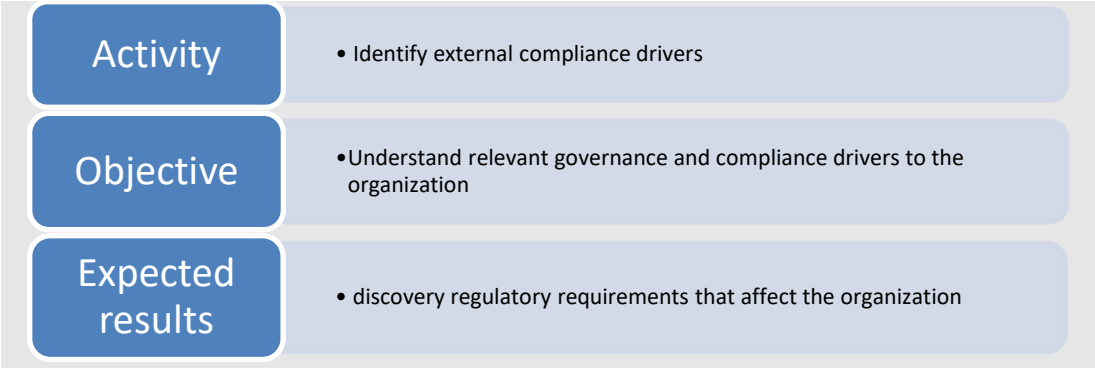

Figure 21 - Policy and compliance activities, objective and expected results in phase two

During phase three and phase four, a list of software requirements from HIPAA compliance driver with its related "must-have" practices the company should be made. As shown in Figure 22.

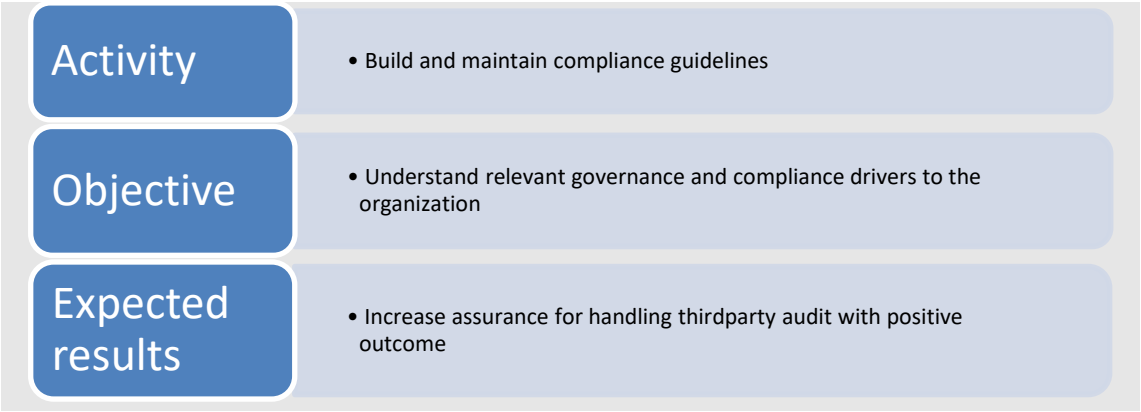

Figure 22 - Policy and compliance activities, objective and expected results in phase four

5.4.3 Education and guidance

Programmer's security experience level plays a major role in developing secure software(Rindell, Hyrynsalmi and Leppänen, 2015). Besides, it is considered an agilic practice (Keramati and Mirian-Hosseinabadi, 2008).

To get the maximum benefit from security practices, security guidance, and raise awareness among the staff about secure programming and best practices is essential.

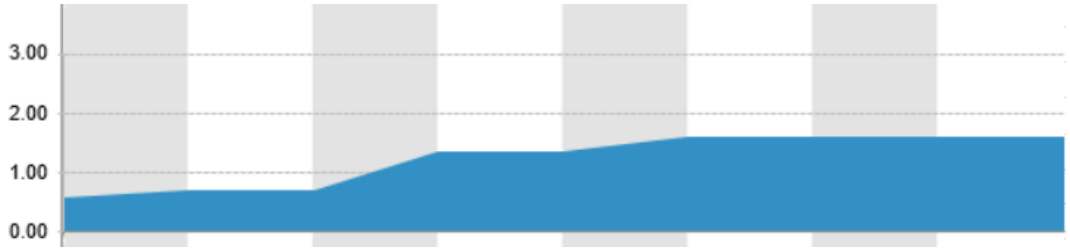

Figure 23 - Progress in education and guidance within four phases

Hence, a collaborative effort should be made to increase awareness about common insecure practices. Starting with security training in the first phase, as indicated in Figure 24 and releasing a security best practices checklist in phase two, as illustrated in Figure 25.

Figure 24 - Education & guidance activities, objective and expected results in phase one

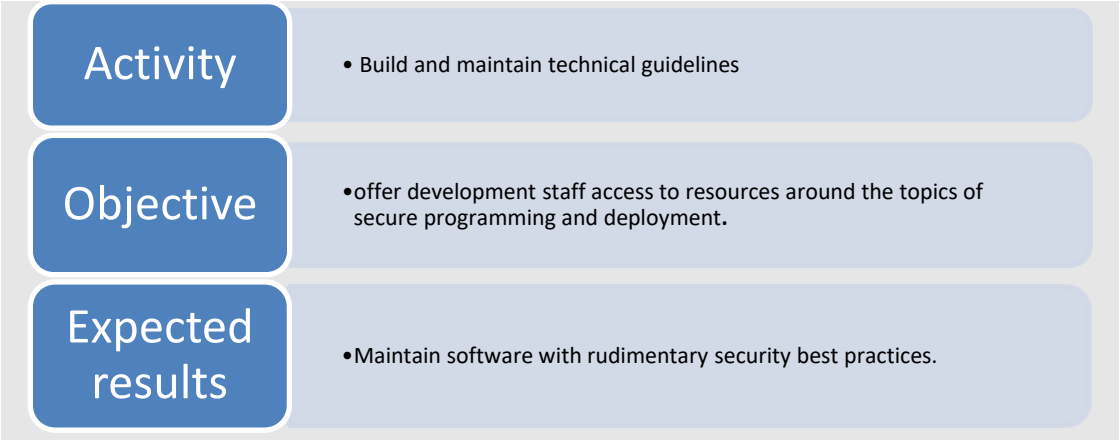

Figure 25 - Education and guidance activities, objectives and expected results in phase two

The target in phase three is building a centralized repository like a website or a portal. In this website, stakeholders can find best-practices lists, FAQs, security-related essays, wiki-based web pages where the staff can communicate and discuss security topics. During phase four, no additional activities are planned for this practice.

5.4.4 Threat assessment

The main point beyond this practice is identifying and understanding threats in an early stage of development lifecycle, putting security and development on the same page, and more importantly, reduce the flaws in system design.
Thus, even when this practice is semi-automated, working on threat assessment is important in Msg Life organization since dealing with threats is considered in an ad-hoc manner, with scarce documentation, and the load on agile is considered acceptable with utilizing DevSecOps tools. The progress in this practice is presented in Figure 26.

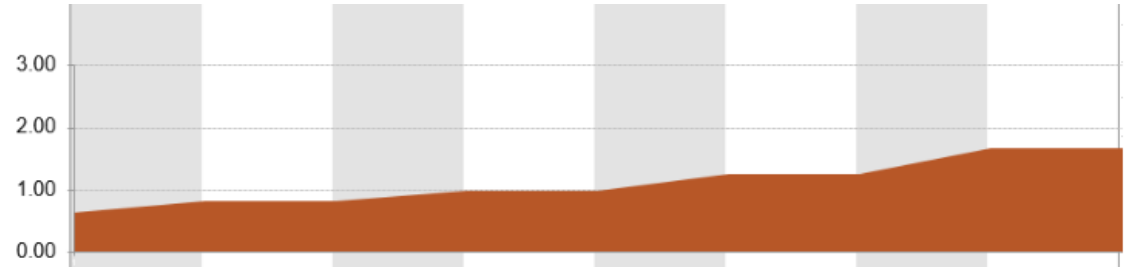

Figure 26 - Progress in threat assessment within four phases

To put this in action, Microsoft threat modeling tool was used to create an initial threat model to the most critical customers project resulted after analyzing the risk profile.

As mentioned in chapter 2 section 2.6.3, this tool depends on Microsoft STRIDE model.

In addition to establishing an attacker profile to have a better understanding of who our enemies might be and their motivations. As explained in Figure 27.

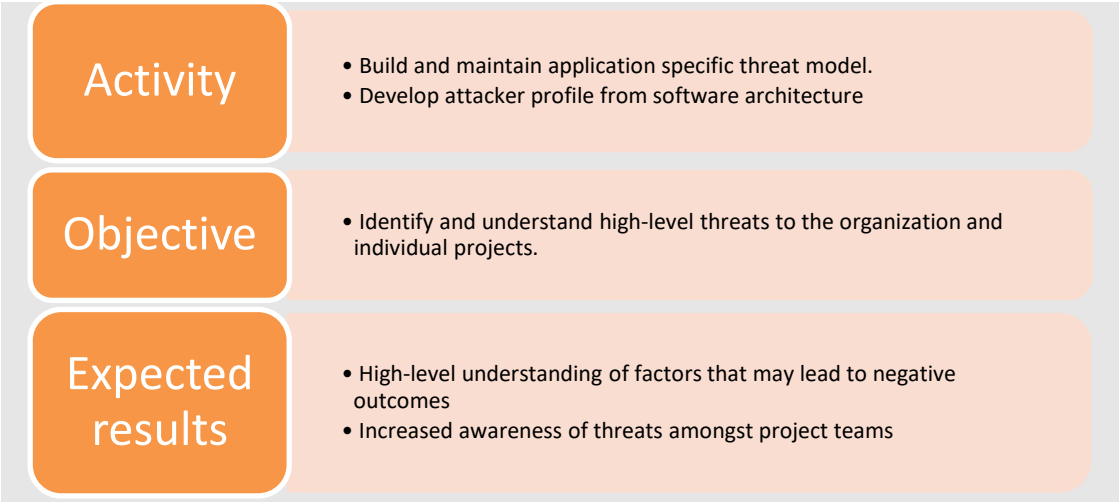

Figure 27 - Threat assessment activities, objectives and expected results in phase one

In the second phase, as represented in Figure 28, special attention was given to third party libraries and dependencies.

To improve the manual way of defining the dependencies, Dependency Tracker is introduced to track external dependencies through using Bill-Of-Material, as clarified in section2.6.6.

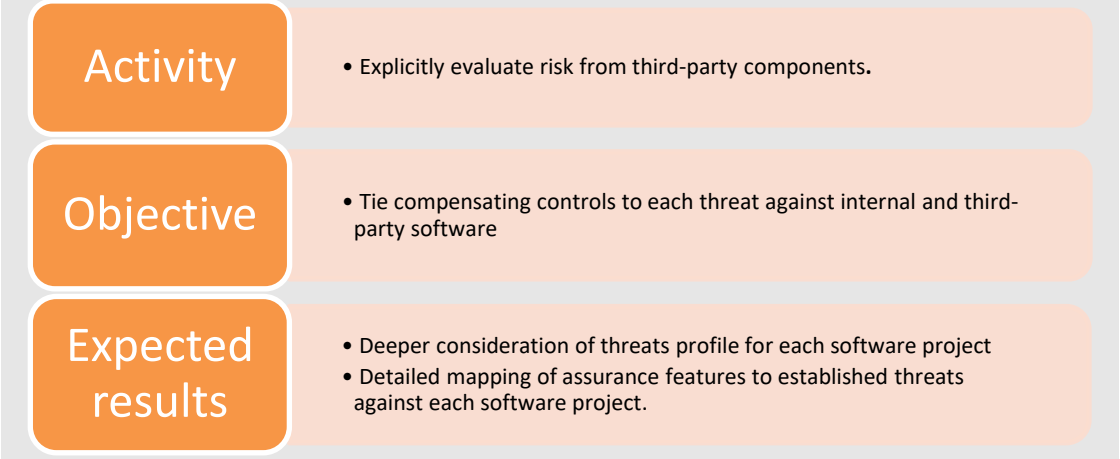

Figure 28 - Threat assessment activities, objectives and expected results in phase two

Figure 29 shows Dependency Track and its ecosystem, where it is provided by the BOM files with CycloneDX format -the recommended one for Dependency Track-, and it uses SonaType OSS Index and NVD vulnerability databases to analyze the dependencies and send the resulting report to the CI/CD tool like Jenkins.

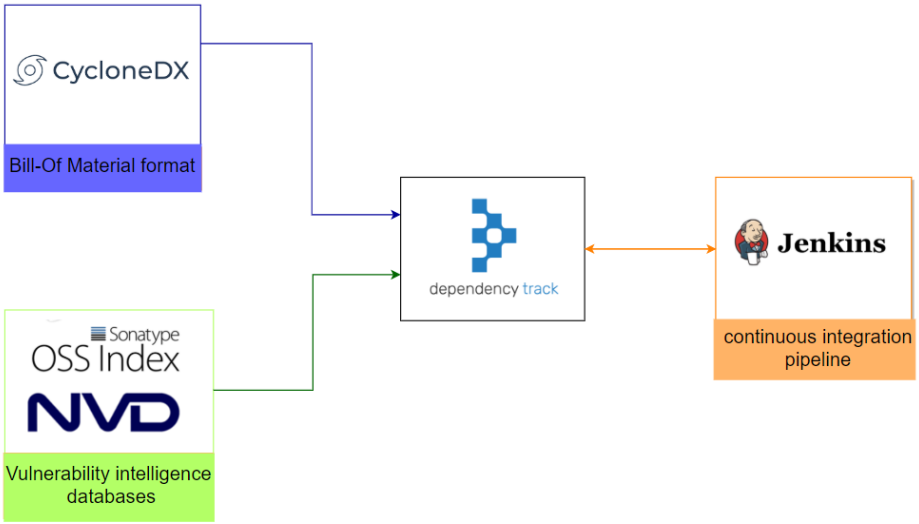

Figure 29 - Dependency Track integrations

As an alternative of the Dependency Track, there is another solution that could be implemented here which is the Dependency Check. But after reviewing each one capability, the Dependency Track was the optimal choice, because it provides further information that interests the company the most, like tracking component usage across all version of every application, and License risk.

Similarly, there are many other tools and methodologies to build the threat model instead of following STRIDE model like creating an attack tree or Trike, and they all work fine. the only preference is that STRIDE is known for being a mature way to develop a threat model. Besides

that, Microsoft threat modeling tool makes things easier especially for other staff who are not so familiar with security.

For phases 3 and 4, the abuse cases will be avoided due to its load on the agile process. Instead, the focus is to create a specific threat model for more projects within the scope, cover half of them in phase three, and the majority of them in phase four.

5.4.5 Security requirements

It is necessary to put robust and specific security requirements to be able to justify our claims later. The security requirement in the Msg Life company is doing fine for the time being, in comparison with the BSIMM. So, slight progress is planned in the second phase of the security roadmap which is enough to enhance this function, as shown in Figure 30.
During phase two, a shortlist of industry best-practices that project teams should treat as requirements is planned. As described in Figure 31.

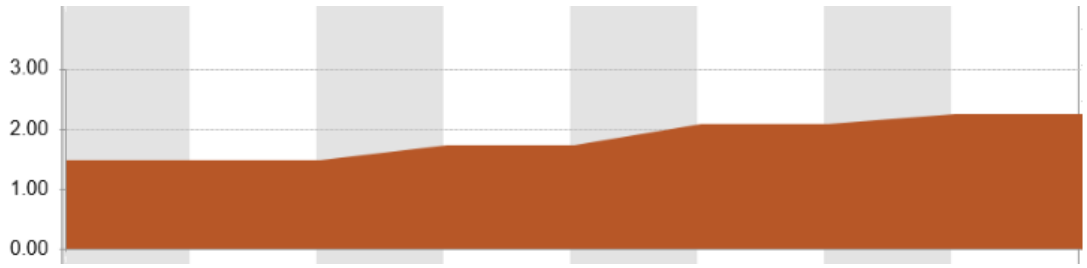

Figure 30 - Progress in security requirement within four phases

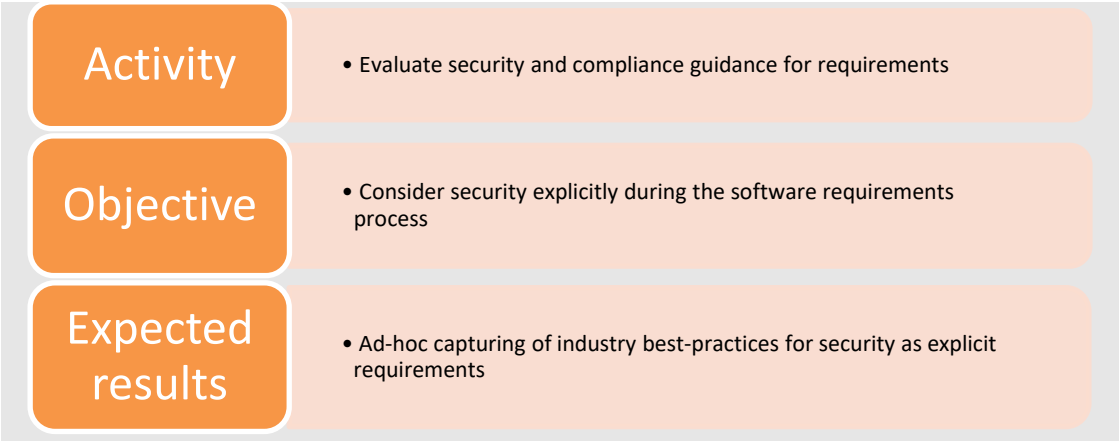

Figure 31 - Security requirement activities, objectives and expected results in phase two

During phase three and 4, the activities will be performed toward integrating the best practices into the requirement phase within the development lifecycle.

Since the security infrastructure like threat models, findings from design review, static code analysis, security testing, etc. will be established, they are going to be able to give some feedback to enhance the requirement phase and make the requirements more security-oriented.

5.4.6 Secure architecture

A slight enhancement in this practice is planned for phase one resulted as shown in Figure 32.

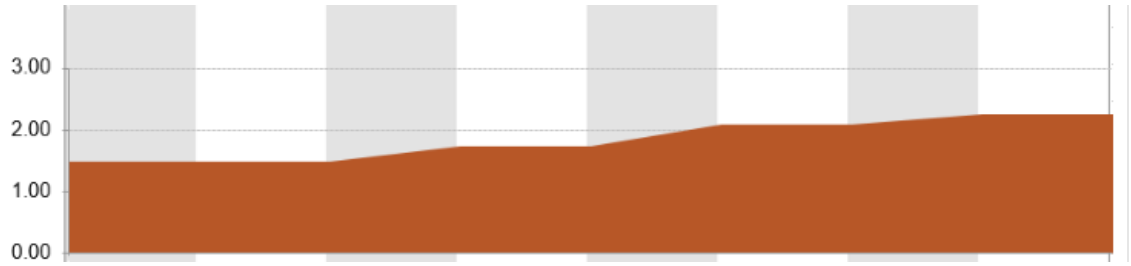

Figure 32 - Progress in secure architecture within four phases

A short checklist of secure design principles like defense in depth, securing the weakest link, use of secure defaults, simplicity in design of security functionality, secure failure, least privilege, avoidance of security by obscurity, etc. was published to the staff as part of the solution, to be used during the design phase in the development lifecycle. As shown in Figure 33 which represents the activity, objective, and expected results for this activity.

On the other hand, using Bill-Of-Material concept and integrating dependency track in the company SSDLC as mentioned before in 5.4.4 the threat assessment practice, to identify common third-party components, boosts security in the architecture and make it easier for the architects to evaluate and weight the reliance on external components in the system.

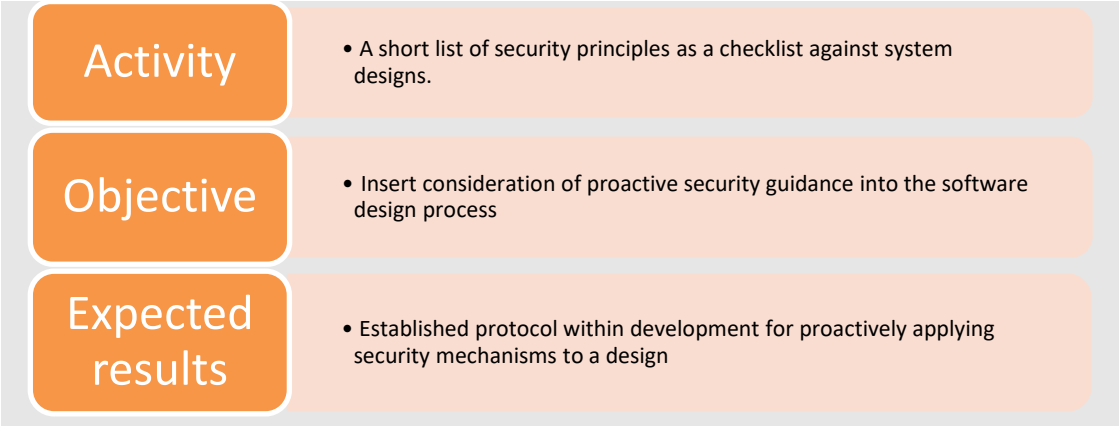

Figure 33 - Secure architecture activities, objectives and expected results in phase two

In phase three, as planned, an increasing number of teams and developers will be informed and complied with secure design practices.

5.4.7 Design review

The main focus of this practice is attack surface analysis and reviewing system architecture from a security viewpoint prior to implementation. Where the architecture is decomposed into smaller components, to set the trust boundaries between these parts and identify entry and exit points, and data flaw in the application. In addition to reviewing the access control for each of these components. As described in 2.6.2.

There is a recursive relationship between attack surface analysis and application threat model since changes in the Attack Surface trigger changes in the threat model including session management, authentication, and password management which directly affect the attack surface. However, design review is considered an important but unautomated security practice.

Given the above, the planned progress in maturity level is demonstrated in Figure 34.

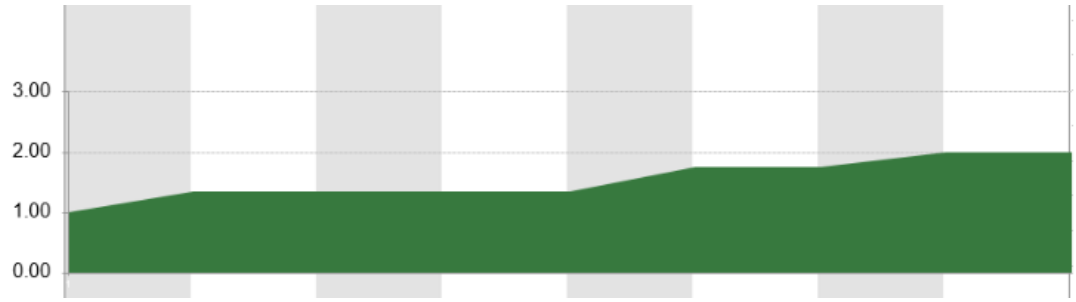

Figure 34 - Progress in design review within four phases

Hence, in phase one, a baseline description of the attack surface exists to identify different entry and exit points that can be used to interact with the system deliberately, unintentionally, or maliciously. In our case, the most important points are APIs, user interface (UI) forms, HTTP headers and cookies, API endpoints, Files, and Databases.

Thereafter, analyzing attack surface can abbreviate the test efforts because It is considered as a pre-seed to the automated test.

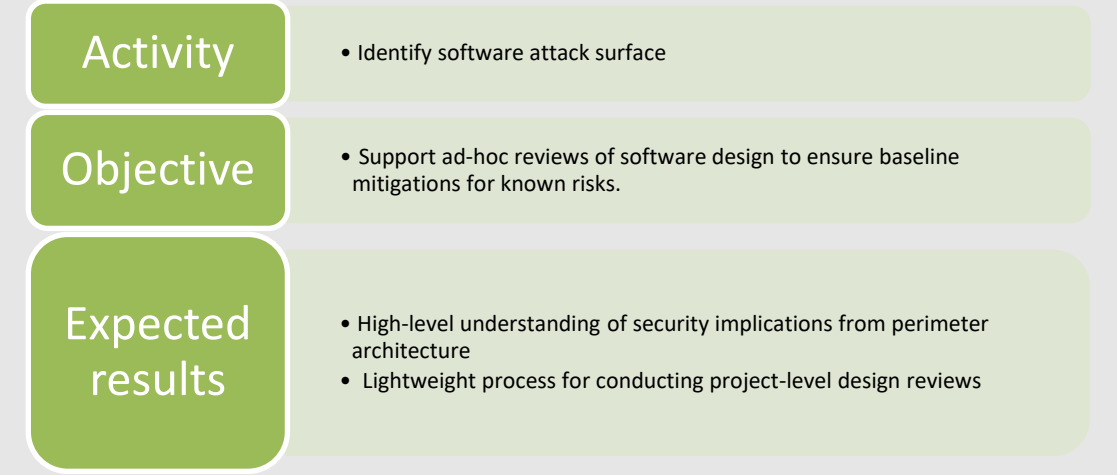

Figure 35 - Design review activities, objectives and expected results in phase one

In phase two, there is no planned change in this practice. But in phases three and four, the number of projects having their attack surface analyzed should increase to reach half of the projects defined in the scope.

Also, in phase three there should be quality gates for design, by setting a particular point in the software development lifecycle where a project cannot pass until a design review is conducted and findings are reviewed and accepted.

5.4.8 Implementation review

As discussed before in section 5.2 in Msg Life there is an initial static code analysis to highlight potential vulnerabilities in the source code using the SonarQube tool that is described in section 2.6.4. Nevertheless, the static code analysis is not fully considered among the development team.

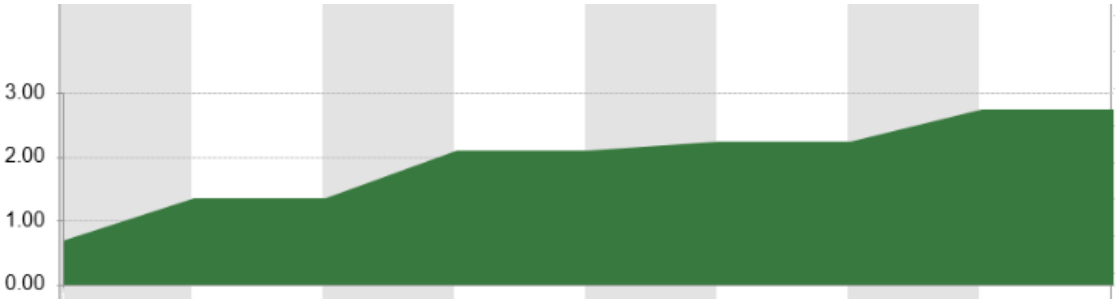

Figure 36 - Progress in implementation review within four phases

As part of the security assurance roadmap, the static code analysis is integrated with the CI/CD process and activated from phase one. Consequently, the source code will be tested after each commit/push.

In Figure 37 the activity, objective, and expected results are presented.

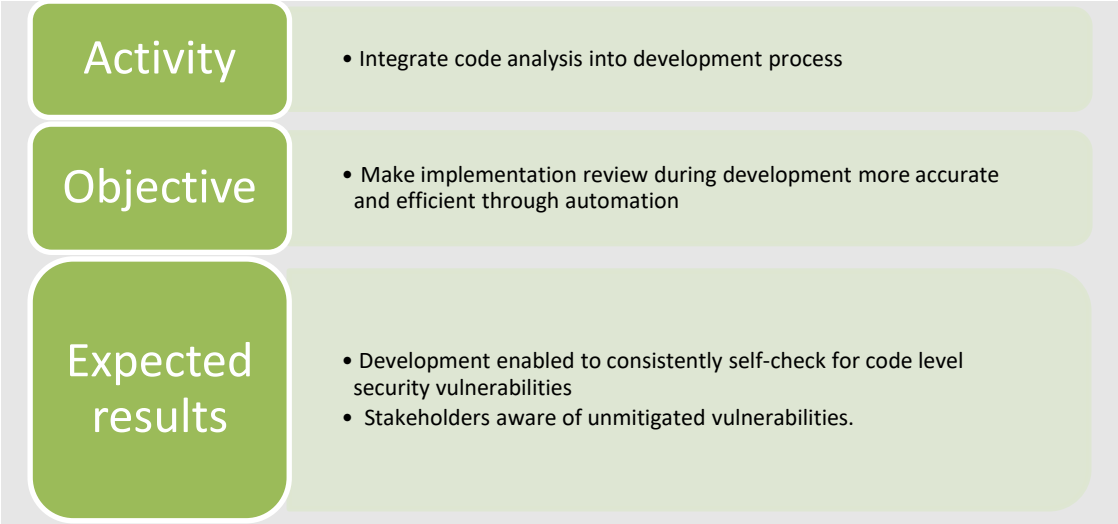

Figure 37 - Implementation review activities, objectives and expected results in phase one

In phase two, the quality gates of the code review will be activated to provide feedback on the discovered vulnerabilities and prevent any critical ones to be in the production environment. As represented in Figure 38.

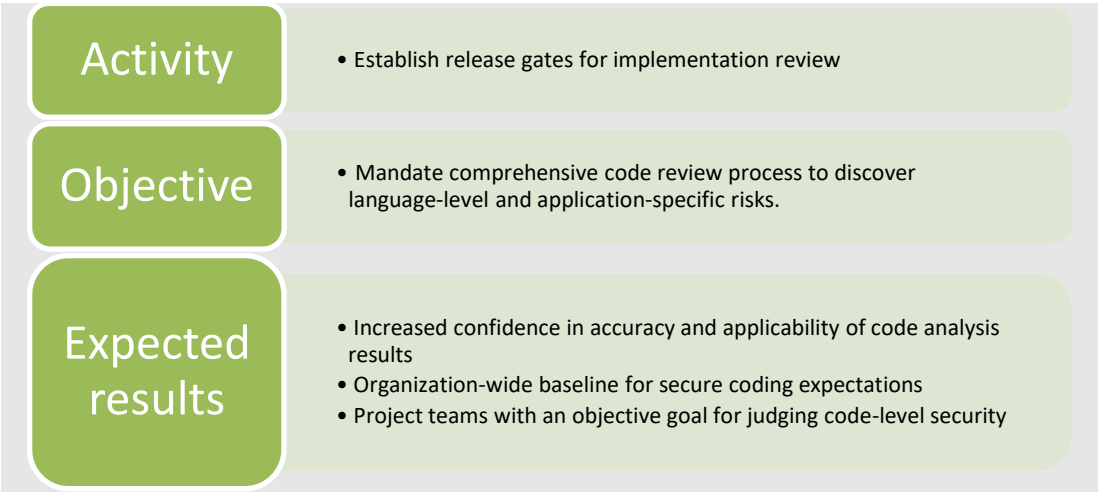

Figure 38 -Implementation review activities, objectives and expected results in phase two

Since critical vulnerabilities are going to be addressed because of this practice. Some security fixes in the source code for the core system should be done even if this change cannot be measured by SAMM project directly. More details in section 5.5.
For phase three, there are no changes in this practice. But in phase four, all the developers and other stakeholders should be more aware of the static code analysis results and it must be reviewed constantly by them. Also, create a sub-plan to address findings in legacy code.

5.4.9 Security test

It is the practice of testing the web application to find exploitable vulnerabilities. These vulnerabilities may exist in operating systems, application flaws, missed configurations, or risky end-user behavior. There are two types of security tests, manual security tests like penetration testing or automated security test that is considered among agilic practice. (Felderer *et al.*, 2016)

Normally, the security test has two main modes, the first one is the active test, where the test is trying to perform actual attacks that involve interactions. On the other hand, the passive test, which is more like observing the system with no direct interaction. (Scarfone *et al.*, 2008)

To integrate the security tests in the CI/CD process, we need to implement the automated security test. For our solution, OWASP ZAP is used because it is an opensource tool and satisfy our needs, other options like Burp suit are considered as an alternative. (*Vulnerability Scanning Tools | OWASP*, no date)

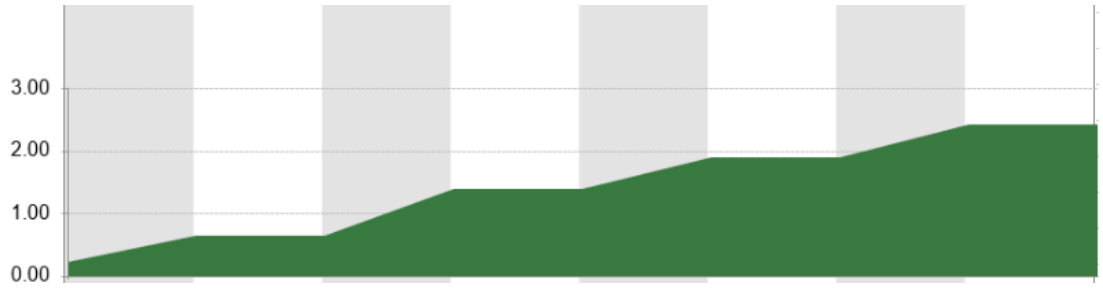

Figure 39 - Progress in security test within four phases

The first phase of the security assurance roadmap includes preparing a generic automated test. And integrate this automated test with the CI/CD pipeline. Figure 40 shows the objective and expected results of this activity.

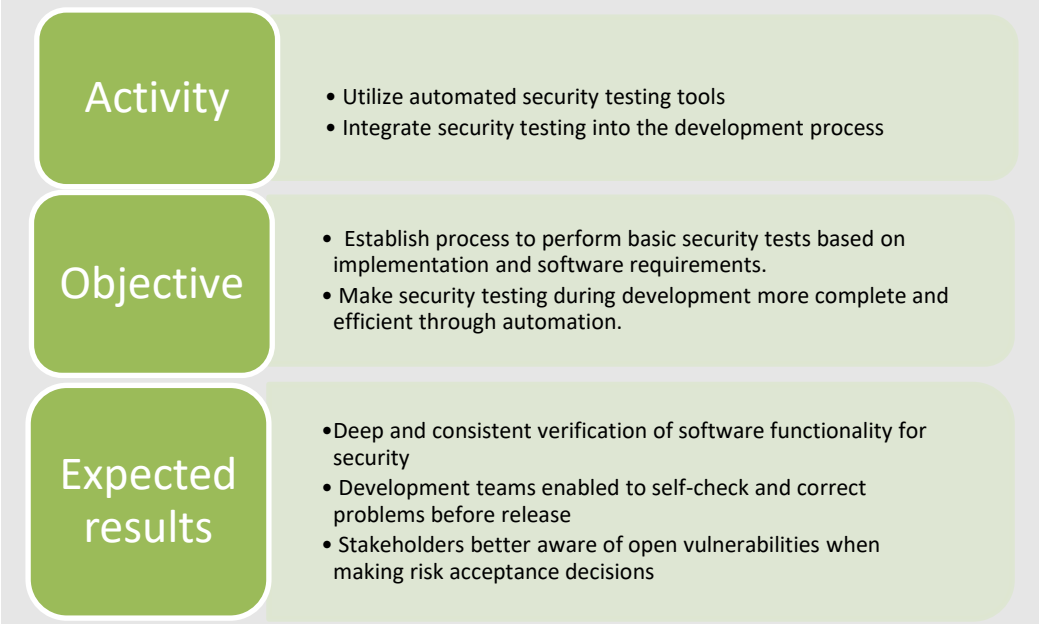

Figure 40 - Security test activities, objectives and expected results in phase one

The second phase of the roadmap includes a customized automated security test to meet each artifact logic, allowing to activate the release gates with each build and keep the staff updated about the vulnerabilities in the code. As shown in Figure 41.

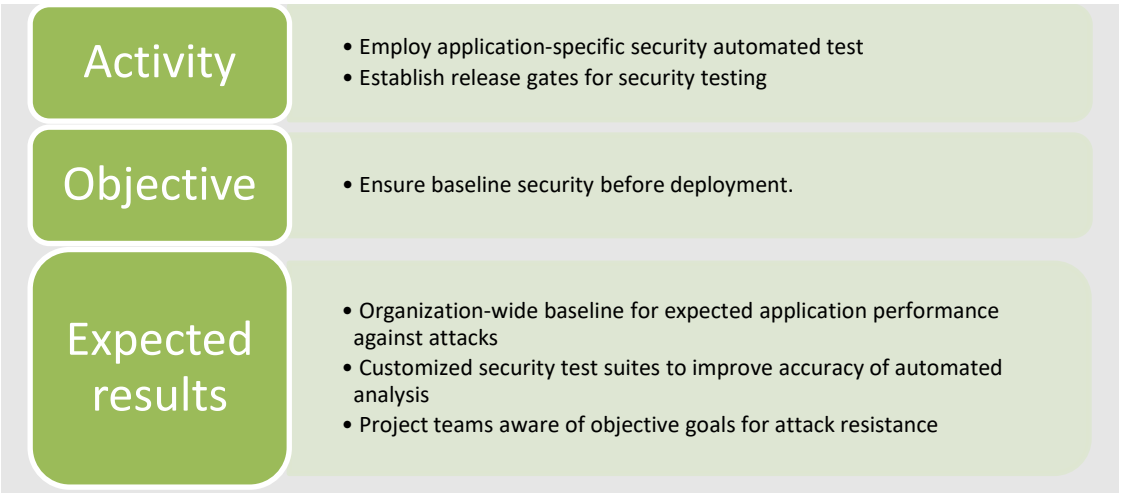

Figure 41 - Security test activities, objectives and expected results in phase two

Phase three targets raising the awareness among stakeholders about the automated test results, document general test cases based on security requirements and common vulnerabilities, increase the number of projects covered by the security test to reach all the projects within the scope and customize the automated test per customer project according to its specific logic.

During phase four, perform penetration testing on critical parts of the system prior to release, is planned and increase the awareness among the development team so all of them are aware of the security test results.

5.4.10 Issue management

This practice's main focus is the incident response plan aims to identify, respond to, and recover from security incidents to prevent damages like service outage, data loss or theft, and illicit access to organizational systems.

Here, the company already has a known point of contact in case of security issues which clarify the result in Figure 42, but there is no incident response plan in place. Thus, the second phase of the security roadmap guarantees designating the point of contact and his surrogate besides publishing an initial plan. As described in Figure 43.

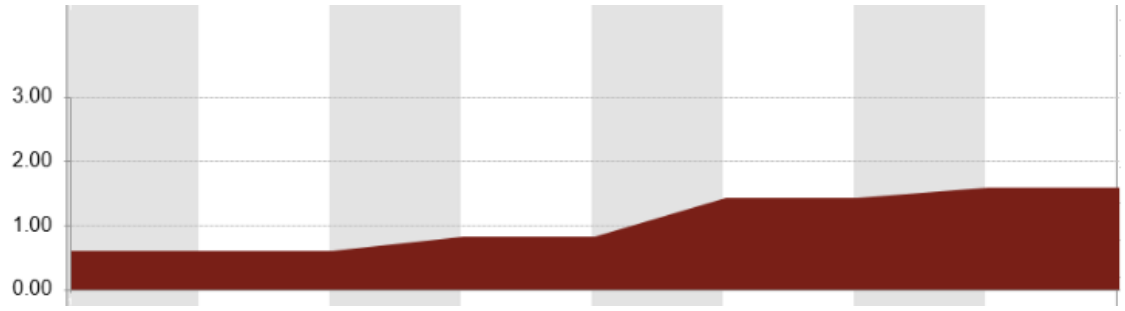

Figure 42 - Progress in issue management within four phases

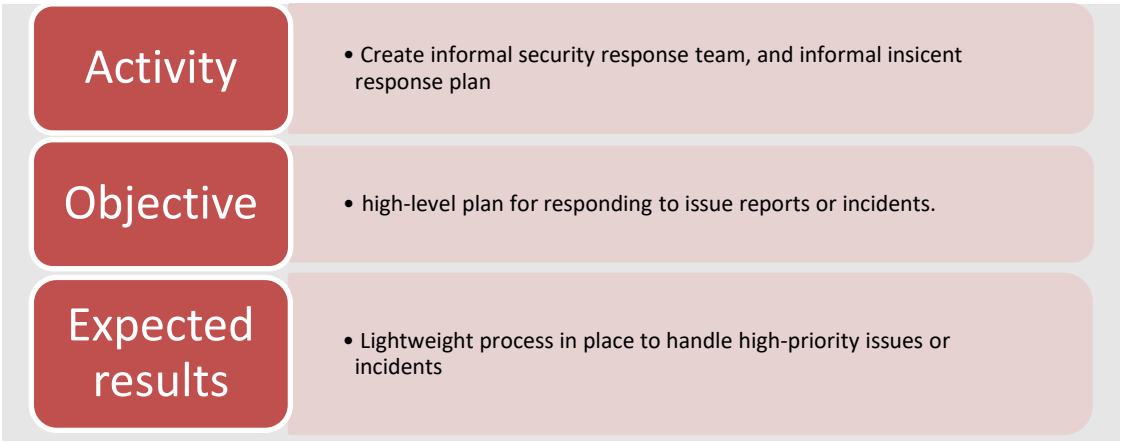

Figure 43 - issue management activities, objectives and expected results in phase two

In phase three, the main exertion in this practice is dedicated to increasing the robustness of the incident response plan by document the organization's incident response process as well as the procedures that team members are expected to follow.
Plus, setting up a consistent security issue disclosure process to communicate with costumers, in addition, to release patches since software is being operated by parties external to the

organization. That's as planned will be limited to the most critical business areas before extending that to be across the company during phase four.

5.4.11 Environment hardening

As mentioned earlier, the operation environment is out of the scope. So, a limited and late enhancement was given to this security practice during phases three and four. As presented in Figure 44.

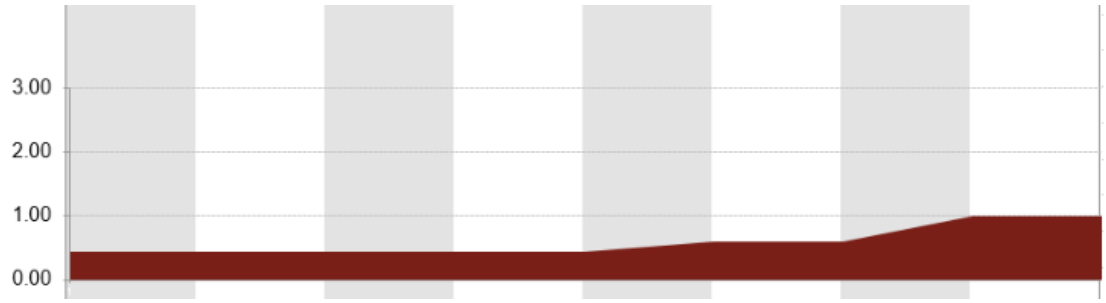

Figure 44 - Progress in environment hardening within four phases

During phase three, as displayed in Figure 45, the definition of the expected operating platforms should be created and maintained. This specification should be jointly created with development staff, stakeholders, support, and operations team, etc.
During phase four, the security team or operations team reviews optional tools for protecting software with project stakeholders like firewalls, IPS…etc.

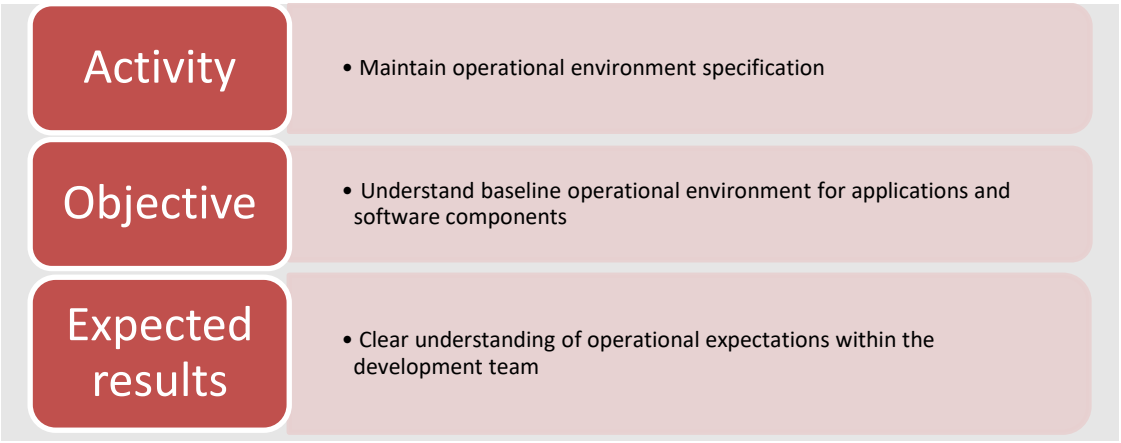

Figure 45 - environment hardening activities, objectives and expected results in phase three

5.4.12 Operational enablement

Similar to previous security practices, operational enablement is considered out of the thesis scope. But a limited advancement could be made during phase three and four.

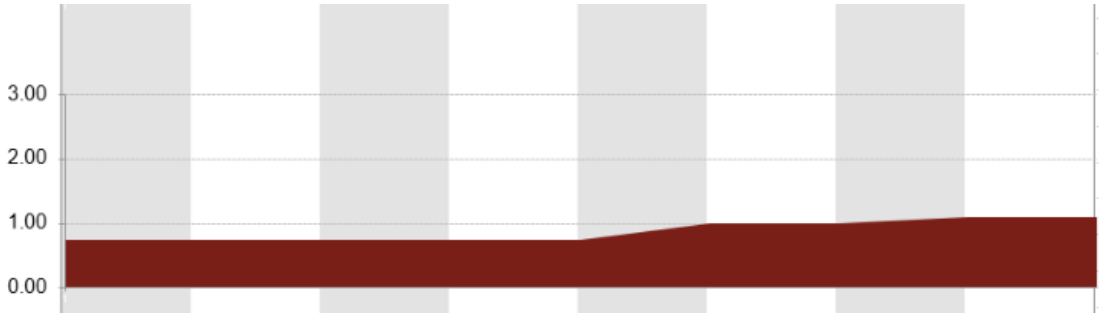

Figure 46 - Progress in operational enablement within four phases

Preparing documentation for the most important error and alert messages which require the user and the operator attention and how to respond to that.
Later, during the fourth phase, this should turn into a guide that contains security-related configuration options, event handling procedures, installation and upgrades guides, operational environment specifications, security-related assumptions about the deployment environment, etc. As described in Figure 47.

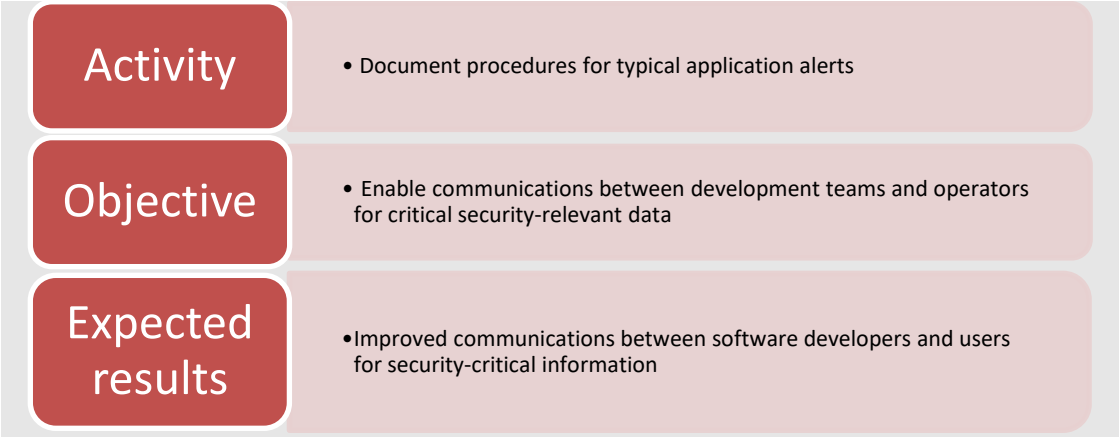

Figure 47 - operational enablement activities, objectives and expected results in phase three

5.5 Phase one - Implementing phase one specified activities

In the previous section, we described in detail a set of activities that need to be implemented to achieve our security goals. In this section, the implementation process of those activities during the first phase of the assurance roadmap is revealed.

Starting with **strategy and metrics,** a list of worst-case scenarios has been established as described in section 5.4.1. and basic risk assessment was made to prioritize projects with the highest risk to cover first by the roadmap.

In **education and guidance**, a list of courses that require approximately one day per developer on secure programming and security best practices were proposed to the company management to be planned and scheduled. Based on the plan in section5.4.3.

In **threat assessment**, Microsoft threat modeling tool was used to create an initial threat model to the most critical system that was concluded from analyzing the business risk profile. As result, the threat model was established for the core product in the company. This tool depends on Microsoft STRIDE. And a more than 90 threat was identified including network failure, erroneous input, SQL injection, pretending of wrong identity, unauthorized usage, data manipulation…Etc.

Furthermore, in this practice, an attacker profile was established and published to the company's internal confluence[17] to have a better understanding of who might be our enemies and what could be their motivation. As described in section 5.4.4.

In **design review**, the attack surface was Analyzed based on the threat model, part of this process was documented, resulting in a list of APIs, endpoints, databases...etc. to provide it with appropriate countermeasures later. As mentioned in section5.4.7.

In the **implementation review**, the SonarQube was updated and fully integrated with the Jenkins CI/CD pipeline to perform the static code analysis per commit/push to the Git server for the projects in the scope. As planned in section 5.4.8.

After successfully integrating SonarQube and analyzing the resulting issues from static code analysis, critical vulnerabilities emerged from the core product. Based on that, fixes are delivered to remediate those issues outside SAMM project model.

100% of the vulnerabilities reported by SonarQube from different severity levels (Blocker, critical, major, and minor) in the core product and one customer project, were resolved, and 100% of the security hotspot issues, which are pieces of security-sensitive code that may or may not be a vulnerability, were reviewed as well.

Accordingly, the majority of those vulnerabilities marked as false positive[18] while only 13.1% were considered actual issues. As demonstrated in Figure 48 which shows the security rating is improving over time and the number of security issues before and after that vulnerabilities' remediation process.

[17] Collaborative tool used to help teams cooperate and share knowledge.

[18] A false positive happens when the static code analysis tool raises a red flag of vulnerability when it is not. (*Static Code Analysis | OWASP*, no date)

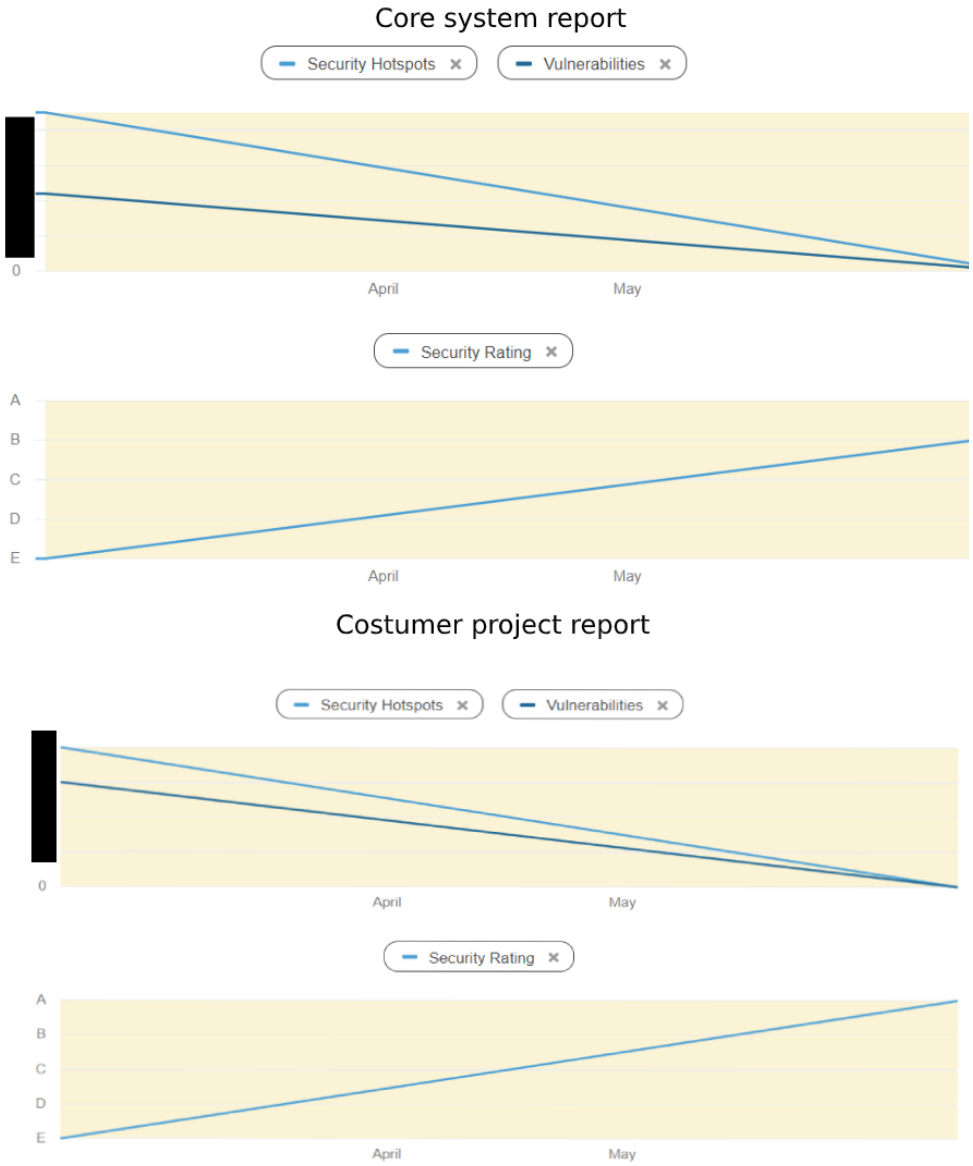

Figure 48 - vulnerabilities report from static code analysis after vulnerabilities' remediation process

In **security test**, OWASP ZAP security scanner was used as mentioned in section 5.4.9 to test the RestAPI for one of the customers' projects, including the services from the core system.

To perform the security test two things are needed, the first one is a valid description of the RestAPI that needs to be tested, plus a valid session to test the endpoints that need authentication and authorization.

in our solution, we used OpenApi[19] specification, which is a machine-readable interface file for describing, producing, consuming, and visualizing RESTful web services, to provide the automated security test a starting point and a Python script was used to get a valid session from the server and attach it to each request that needs authentication.

[19] https://swagger.io/specification/

The automated security test is configured through a configuration file to define a list of security tests to be performed (some of them are passive while the others are active) and integrated with the CI/CD pipeline.

More details of the automated security test are shown in Figure 49.

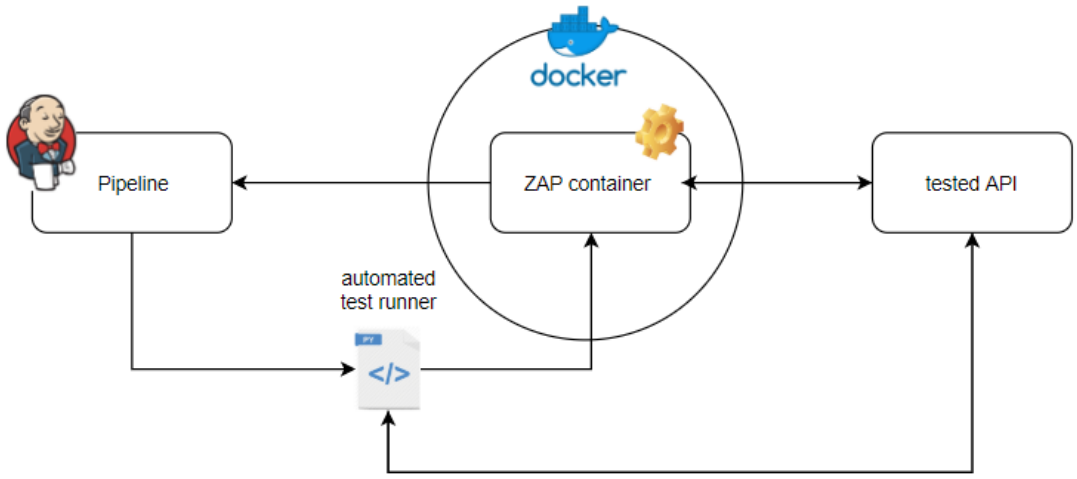

Figure 49 - Automated security test design

Jenkins pipeline triggers the automated security test script. The runner script gets a valid session from the tested RestAPI first to provide OWASP ZAP with it. Using the following command when the body.json is a JSON object contains a valid credential:

```
curl --header "Content-Type: application/json" --request POST  --data @body.json ''' + loginUrl
```

Code 1 - getting valid session for the automated security test

Next, the test script attaches the latest session ID from the previous step and launches OWASP ZAP Docker container providing it the configuration file (config.prop) using the following command:

```
1.  docker run -v "**\**' + projectName + '''":/zap/wrk/:rw owasp/zap2docker-
    weekly zap-api-scan.py -t openApi.yaml -f openapi
2.  -z " -configfile /zap/wrk/config.prop"
3.  -c api-scan.conf -d -r api-scan-report.html''')
```

Code 2 - Running ZAP inside docker container

Once OWASP ZAP finishes its tests on the targeted RestAPI, it publishes the security test report to Jenkins in HTML formal.

5.6 Phase one - Evaluating results

According to what has been accomplished during this phase, Table 6 draws the conclusion of phase one, where all the planned activities were applied, and the second phase of the security roadmap is ready to set out.

Table 6 - Resulting security maturity values for phase one

Practice	Phase one expected values	Phase1 result
Strategy & metrics	0.33	0.33
Policy & Compliance	0.20	0.20
Education & Guidance	0.70	0.70
Threat Assessment	0.83	0.83
Security Requirements	1.50	1.50
Secure Architecture	1.85	1.85
Design Review	1.35	1.35
Implementation Review	1.35	1.35
Security Testing	0.52	0.52
Issue Management	0.60	0.60
Environment Hardening	0.45	0.45
Operational Enablement	0.75	0.75

5.7 Phase two – Roadmap update

Upon terminating phase one in the roadmap, the second phase should be initiated by streamlining the roadmap if needed. Therefore, a few changes were made to improve the roadmap with more accurate and consistent targets.

In the **Policy and compliance**, considering the compliance requirements for the third party regulatory does not seem so realistic activity to do during the discussed one-year roadmap. So, it was avoided, changing the policy and compliance maturity values for phase three and phase four to 0.60 and 0.85 consecutively.

In the **design review** practice, the design quality gates seem to be an extra and unnecessary load for agile process. Based on that, the decision was to avoid it, making the target maturity value for this practice 1.35 for phase three and 1.50 for the fourth phase.

Similarly, in the **implementation review,** as planned on the initial roadmap, the organization will derive a code review checklist based on security requirements. Since this activity is a manual activity that could slow down agile and the static code analysis is placed, it was excluded from the roadmap. By changing that, the targets for implementation review for phases two, three, and four are 2, 2, 2.5 consecutively.

In **security test,** the corresponding maturity values for phase three and four to 1.68 and 2.08 where changed as well because documented test cases for security requirements, is a time-consuming process to be done during the development lifecycle, so it was replaced by the ad-hoc test cases setting approach. also, the penetration test is postponed to be implemented later.

5.8 Phase two – Implementing planned activities

In this section, we have a short description of how the activities that are already defined in the previous stage was implemented to achieve the security goals.

In **strategy and metrics** practice, improving the roadmap to include more projects from the scope was accomplished. Also, OWASP Risk Rating Methodology was used to estimate the overall security risks and its related document was published to the internal company's Confluence after reviewing, to be viewed by stakeholders. According to section 5.4.1.

In **policy and compliance**, HIPAA informal compliance checklist which composes of general questions about the measures the organization should have was not fulfilled by the company as described in section 5.4.2.

In **education and guidance**, a checklist of security best practices based on OWASP secure coding best practices[20], Oracle security guidelines[21], and prioritized based on the results from static code analysis to be more oriented to the actual development issues. Finally. It was published to the company's Confluence where it can be accessed by developers. As previously planned in5.4.3 section.

In **threat assessment**, to evaluate threats from third-party dependencies, Dependency Tracker which is one of DevSecOps tools provided by OWASP and described previously in chapter 2 section 2.6.6 to track external dependencies and how each threat and vulnerability in the external dependencies affect the final product. dependency Track tool was integrated with the continuous integration pipeline to give immediate, accurate and per build results regarding the external dependencies. As discussed in section 5.4.4.

It worth mentioning here, Dependency Track depends on Bill-Of-Material, so Bill-Of-Material was introduced to the company prior to implementing it and CycloneDx is used for this purpose.

Since the Dependency Track does not support multiple BOM description for one project, and the company uses Java programming language and Maven as a building tool, besides using JavaScript for the frontend part, the BOM needs to be generated separately for the frontend and the backend and merged afterward. A special script to do this functionality is attached in Appendix E Code 7.

Figure 50 contains four charts shows the increased number of projects covered by Dependency Track, the auditing progress, the number of dependencies, and components that are used in those projects.

[20] https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/
[21] https://www.oracle.com/java/technologies/javase/seccodeguide.html

Figure 50 - dependency track implementation

In the **security requirement,** the best-practices were not published to the company as discussed in 5.4.5. So, the target of this practice was not met.

In **securing the architecture**, a secure design principles checklist was published to the Confluence of the company. So, it can be accessed and viewed by the stakeholders. The main resource of those principles was CERT official website[22]. According to the plan in section 5.4.6.

Also, to enrich the design guidance documentation, a list of API secure design principles was established based on OWASP top 10 for APIs[23], which was published in 2019.

In the **implementation review**, a new capability was added to SonarQube to send a webhook to the Jenkins pipeline. Where the new arisen vulnerabilities are reported back to the developer. As previously mentioned in 5.4.8.

In the **security test**, more developers are aware of the automated security test, and its generated report through presenting a use case to the company. Also, a baseline of quality gates is established. The quality gates could fail the CI/CD pipeline whenever a critical issue was found before reporting that back to the developer via Slack, the communication platform that is widely used in the company. As proposed in section 5.4.9.

In **issue management**, a baseline of the incident response plan was established and published to stakeholders in the company's internal Confluence.

The main steps are described based on CREST research project incident response guide. Which divides the process into 3 main phases, starting with Preparing for a cybersecurity incident by performing a criticality assessment. And responding to a cybersecurity incident, which means identifying the cybersecurity incident and taking appropriate action and recovering from a cybersecurity incident. Finally, the follow-up phase, that includes reporting the incident to relevant consumers and stakeholders, post-incident review, and the learned lessons. (Creasy, 2013)

[22] https://www.us-cert.gov/
[23] https://owasp.org/www-project-api-security/

5.9 Phase two - Evaluating the results

Based on the implemented activities in this practice, Table 7 presents the maturity values improvement during this phase. Where almost all the targeted values are achieved. Except for the targeted value in the security requirements and the targeted value in the security test was not fully achieved.

Table 7 - Resulting security maturity values for phase two

Practice	Phase two expected values	Phase2 result
Strategy & metrics	0.60	0.60
Policy & Compliance	0.35	0.20
Education & Guidance	1.35	1.35
Threat Assessment	1.00	1.00
Security Requirements	1.75	1.50
Secure Architecture	2.00	2.00
Design Review	1.60	1.60
Implementation Review	2.10	2.10
Security Testing	1.27	1.17
Issue Management	0.83	0.93
Environment Hardening	0.45	0.45
Operational Enablement	0.75	0.85

5.10 CI/CD pipeline design and implementation

To decrease the load on agile process, the CI/CD scripted pipeline is designed for the core product in the company as shown in Figure C 1 in the appendix, to achieve the maximum automation possible.

The pipeline is triggered, per commit/push to the Git server, to pull the changes and generate the BOM files that describe the dependencies for the subprojects in parallel. As Code 5 shows. These BOM files are merged to be uploaded to Dependency Track in Code 7.

Dependency Track Analyzes the dependencies and reports them back to Jenkins to check the quality gates. After that, the SonarQube is launched to do the static code analysis and report back to Jenkins if the test passes the quality gates or not. As clarified in Code 4.

Based on that, the pipeline status is reported to Slack. Followed by launching all costumers' projects that depend on the core product.

Each customer project runs its pipeline in a way similar to the one for the core product. As illustrated in Figure C 2 in the appendix. After triggering the pipeline, it is expected to check

the project dependencies using Dependency Track, check the dependency track quality gates. Then, perform the static code analysis in SonarQube and check its quality gates.
Followed by doing the automated security test and publish its report. Finally, report back to the developer.

5.11 Security practices in the development lifecycle

In Table 8 below the implemented security activities are summarized alongside their security practice and development phase.

Some of the activities' development phases were not mentioned explicitly in SAMM project guide. So, it was concluded from other resources like (Nunez, Lindo and Rodriguez, 2020) and MS-SDL model for agile.

Table 8 - Security activities and their development phases

Security practice	Security activity	Development phase
Strategy & metrics	• Build and maintain a security assurance program roadmap within the defined scope. • maintain the assurance program roadmap.	transversal through all the lifecycle
	• Estimate the overall business risk profile. • Update business classification based on business risk	Requirements
Policy & Compliance	• Identify external compliance drivers	Transversal through all the lifecycle
Education & Guidance	• Technical security awareness • Technical guidelines	Transversal through all the lifecycle
Threat Assessment	• Build and maintain an application-specific threat model. • Develop an attacker profile from software architecture • Explicitly evaluate risk from third-party components.	Requirements and design
Security Requirements	• Evaluate security and compliance guidance for requirements	Requirements
Secure Architecture	• Apply secure design principles	Design
Design Review	• Identify software attack surface	Design
Implementation Review	• Integrate code analysis into the development process • Check static code analysis release gates	Implementation
Security Testing	• Utilize automated security testing tool that is integrated into the development process • Check the release gates for security tests	Implementation and testing

Issue Management	• Incident response plan	Post-release
Environment Hardening	• Maintain operational environment specification	Post-release
Operational Enablement	• Document procedures for typical application alerts	Implementation and post-release

5.12 Conclusion

Throughout this chapter, the first and second phases are fully fulfilled, including their substages. Accordingly, the underlying security activities were explained. The implementation of the specified activities that included in the first and second phases was also achieved.

Hence, the maturity level after those two phases was changed, in the following chapter, we demonstrate more details about those outcomes.

6 Experimentation and Evaluation of the Solution

In computer science we build new systems, find new techniques and sometimes we establish new trends, sometimes it is obvious when we achieved our goals and satisfied our objectives and how to evaluate that. But, in the information security field, it is a hazy operation to measure the impact explicitly because we are working on preventing bad actions.

In this chapter, we clarified the hypothesis, the indicators, and how we evaluated our work in our security assurance roadmap.

6.1 Investigation hypothesis specification

Adding security assurance practices to the existing SDLC within the organization, using the proper methodology, and considering security as a continuous concern in each phase during the development by shifting-left security to an early phase could significantly increase the security maturity level in the company and decrease the cost of fixing flaws and vulnerabilities without overwhelming agile process.

6.2 Identification of indicators and sources of information

Implementing continuous security is a set of activities, so we expect the indicators to be across the organization and its targeted project scope in four different security functions which will be reflected in the maturity test result after each phase of the roadmap in SAMM project.

The indicators and the source of information in the maturity context for the two implemented phases can be summarized into the four main functions of SAMM project like the following:

In the governance function: the existence of the security assurance strategy, setting up the developers' secure programming guidelines and training, and a list of the most critical business-level risks.

In the threat assessment function, the indicators can be classified into a continuously updated threat model, attacker profile, and attack surface analysis. High-level understanding of factors that may lead to negative outcomes, increased awareness of threats amongst project teams, and having a consistent and automated process to detect threats from third-party components.

In the verification function: the indicators are the percentage of tested projects using the automated security tests and analyzed using static code analysis and establishing and control these two practices with quality gates. Also, the awareness among the development staff of the security tests is considered an indicator here.

In the operation function, the indicators that can be observed here are the existence of a lightweight process to handle high-priority issues or incidents, and assign the organization point of contact.

The final and explicit source of information is the decreased number in the vulnerabilities and flaws in the final products that are detected by static code analysis and security tests over time, in comparison with the security issues that already discovered in the application.

6.3 Description of the evaluation methodology

Our proposed evaluation methodology is represented by conducting a security maturity evaluation test to assess the situation after implementing continuous security practices into the development lifecycle and compare it with the expected level of security maturity value that specified in the objectives.

SAMM project comes with a robust maturity assessment tools to help to assess the company's situation in each phase along with the roadmap and compare the maturity level for each security practice with the initial value to measure the progress in this practice and compare it with the target values to measure the achievement whether the targets were met or not.

For our work, we established the roadmap for Msg Life for one year. The roadmap consists of four phases. Those four phases contain the core and the most appropriate activities that respects agile process. The first and second phases are implemented in the context of this thesis, while the other two phases are up to the company to commit to them after the internship period.

6.4 Evaluating the results

We discussed earlier in section 5.3, the targeted maturity values for all the roadmap.

In section 5.6, the maturity values resulting from phase one are presented. After some changes in the roadmap as shown in section 5.7, the targeted values in the road map are updated for some security practices. And the results after implementing phase two of the roadmap are presented in section 5.9.

Figure 51 clarifies the results; the figure shows a comparison between the initial roadmap expected maturity values over the four phases and the resulting maturity levels after implementing the first phase (the red area) and the second phase (the area in yellow).

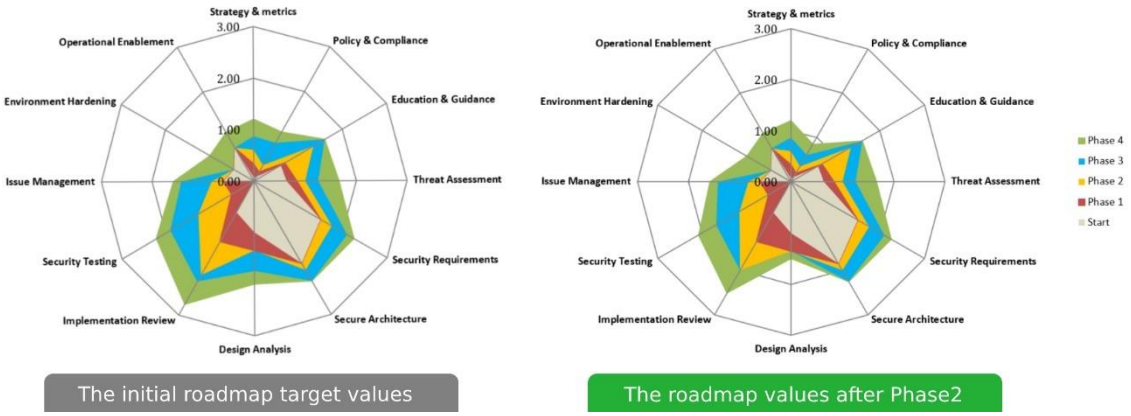

Figure 51 - comparing the roadmap after the second phase with the initial target values

The full tables of the initial security maturity values and the security maturity values after phase two are attached in Table A 1 and Table A 3 in the Appendix.

In view of the foregoing results, the targeted values in the model are almost achieved, the overall security maturity value increased from 0.69 to 1.13 out of 3 and the code-based vulnerabilities has been reviewed and resolved. thus, the objective of this thesis was achieved.

7 Conclusion

7.1 Summary

Software security is a cornerstone of today's software industry. There are various security assurance approaches to integrate it with the development process.

In the context of this thesis, we pursued the goal to establish a secure development process. Hence, many ways to introduce security to agile companies were discussed, and after analyzing many available approaches, we selected OWASP SAMM Framework which is one of currently evolving approaches.

In Msg Life company, we planned for an iterative roadmap, established it for four main phases, and implemented the required activities in the first two phases. To track the result, we used the security maturity evaluation process, which a part of OWASP SAMM.

During phase one, the focus was to develop the initial roadmap, based on a realistic assessment, in addition to establishing the required fundamentals using a carefully selected set of activities across the scope of targeted projects. The number of projects which are within this phase was limited to a small set, prior to spreading them out to cover more projects and involving the stakeholders more in this process in the following phases.

The second phase mainly targeted the verification activities, as well as increasing the maturity level of the activities performed in the previous phase and automate much of the process through the CI/CD pipeline.

Phase three and four aim to enhance the overall security maturity level according to the roadmap. After finalizing the one-year roadmap, the organization can iterate again to further enhance security.

Based on results from the evaluation reporting after the first and second phases, applying SAMM project security assurance roadmap has resulted in more secure software and reduced vulnerabilities findings and the cost stemmed from the risk and from fixing them.

7.2 Future directions

In Msg Life, there are many aspects to further enhance the security posture as described in the roadmap, by performing the defined activities and applying them to the existing development lifecycle.

Moreover, an additional refinement can be done afterward, thanks to the iterative approach of the roadmap (preparation, assessment, identifying the targets, setting the activities, and evaluating the results). A considerable work can be done, particularly under the governance security function concerning compliance with HIPAA regulatory, empowering the development guidance, extending the automated security test, and customizing it.

Additionally, improving the CI/CD pipeline performance could be a bonus since the execution time of the pipeline could significantly increase, leading to less productivity.

Overall, aligning the security activities with Scrum Development Process (pregame phase, game phase, aftergame phase) can lead to smoother integration between agile and security.

In summary, SAMM project showed an acceptable level of flexibility and adaptation in agile. However, the outcome can be far more pragmatic if there is a different version of the framework, that takes the agile nature of the process more into consideration, and measure the effect of each security practice on agile to make sure that it doesn't exceed a given threshold i.e. time spent per sprint, automation level, people involved.

References

2018 Cost of Insider Threats: Global Sponsored by ObserveIT (2018).

2019 Survey on Agility Agi l e Transformati on From Agile experiments to operating model transformation: How do you compare to others? (2019).

Akao, Y. and Mazur, G. H. (2003) 'The leading edge in QFD: Past, present and future', *International Journal of Quality & Reliability Management*. MCB UP Ltd, pp. 20–35. doi: 10.1108/02656710310453791.

Alberts, C. J. *et al.* (1999) *Operationally Critical Threat, Asset, and Vulnerability Evaluation SM (OCTAVE SM) Framework, Version 1.0*.

Attack Surface Analysis Cheat Sheet | OWASP (no date). Available at: https://owasp.org/www-project-cheat-sheets/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet (Accessed: 23 February 2020).

Bedi, P., Gandotra, V. and Singhal, A. (2013) 'Innovative strategies for secure software development', *Software Design and Development: Concepts, Methodologies, Tools, and Applications*, 4–4, pp. 2099–2119. doi: 10.4018/978-1-4666-4301-7.ch097.

Burp Suite - Cybersecurity Software from PortSwigger (no date). Available at: https://portswigger.net/burp (Accessed: 23 February 2020).

Chandra, P. (no date) 'Software Assurance Maturity Model http://www.opensamm.org', *Agenda*.

Code Quality and Security | SonarQube (no date). Available at: https://www.sonarqube.org/ (Accessed: 23 February 2020).

Creasy, J. (2013) 'Cyber Security Incident Response Guide', *Crest*, pp. 1–56.

Davis, N. (2013) 'Secure Software Development Life Cycle Processes', pp. 1–29. Available at: http://resources.sei.cmu.edu/asset_files/whitepaper/2013_019_001_297287.pdf.

Dawson, M. *et al.* (2010) 'Integrating software assurance into the software development life cycle (sdlc) meeting department of defense (dod) demands', *Journal of Information Systems Technology and Planning*, 3(6), pp. 49–53. Available at: http://web.ebscohost.com.library.capella.edu/ehost/pdfviewer/pdfviewer?sid=af3a1ddd-558e-4441-8b04-f936954d83b8@sessionmgr14&vid=2&hid=12.

De, B., Distrinet, W. and Leuven, K. U. (2007) *The OWASP Foundation OWASP Day CLASP, SDL and Touchpoints compared*. Available at: http://creativecommons.org/licenses/by-sa/2.5/http://www.owasp.org/ (Accessed: 17 February 2020).

Discotek.ca Deep Dive (no date). Available at: https://discotek.ca/deepdive.xhtml (Accessed: 23 February 2020).

Felderer, M. *et al.* (2016) 'Security Testing: A Survey', in *Advances in Computers*. Academic Press Inc., pp. 1–51. doi: 10.1016/bs.adcom.2015.11.003.

Find Security Bugs (no date). Available at: https://find-sec-bugs.github.io/ (Accessed: 23 February 2020).

Goertzel, K. and Hamilton, B. A. (2007) *Assurance*. Available at: https://www.researchgate.net/publication/279351339.

Heinonen, K. (2004) 'Reconceptualizing customer perceived value: The value of time and place Customer dominant logic View project Design of Multichannel Services View project', *Article in Journal of Service Theory and Practice*. doi: 10.1108/09604520410528626.

Hermiyanty, Wandira Ayu Bertin, D. S. (2017) *Applied software measurments - Global analysis of productivity and quality*. 3rd edn, *Journal of Chemical Information and Modeling*. 3rd edn.

doi: 10.1017/CBO9781107415324.004.
Ibm (2019) *Cost of a Data Breach Report 2019*.
Jagannathan, V. (no date) *The OWASP Foundation OWASP Threat Modeling Architecting & Designing with Security in Mind*. Available at: http://www.owasp.org (Accessed: 26 June 2020).
Jarzombek, J. (2012) *Software Assurance: Enabling Security and Resilience throughout the Software Lifecycle*.
Kainerstorfer, M., Sametinger, J. and Wiesauer, A. (2011) 'Software security for small development teams: A case study', *ACM International Conference Proceeding Series*, (December 2011), pp. 305–310. doi: 10.1145/2095536.2095590.
Kaur, K., Jajoo, A. and Manisha (2015) 'Applying agile methodologies in industry projects: Benefits and challenges', *Proceedings - 1st International Conference on Computing, Communication, Control and Automation, ICCUBEA 2015*. Institute of Electrical and Electronics Engineers Inc., pp. 832–836. doi: 10.1109/ICCUBEA.2015.166.
Keramati, H. and Mirian-Hosseinabadi, S. H. (2008) 'Integrating software development security activities with agile methodologies', in *AICCSA 08 - 6th IEEE/ACS International Conference on Computer Systems and Applications*, pp. 749–754. doi: 10.1109/AICCSA.2008.4493611.
Khaim, R. *et al.* (2016) 'A Review of Security Integration Technique in Agile Software Development', *International Journal of Software Engineering & Applications*, 7(3), pp. 49–68. doi: 10.5121/ijsea.2016.7304.
Koen, P. *et al.* (2001) 'Providing clarity and a common language to the "fuzzy front end"', *Research Technology Management*. Industrial Research Institute Inc., 44(2), pp. 46–55. doi: 10.1080/08956308.2001.11671418.
Mcdermott, J. and Fox, C. (no date) *Using Abuse Case Models for Security Requirements Analysis*.
McGraw, G. (2006) 'Software Security: Building Security in', in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, p. 6. doi: 10.1109/ISSRE.2006.43.
Microsoft Corporation (2010) 'Microsoft SDL_Version 5', p. http://www.microsoft.com/en-us/download/confirmati. Available at: https://www.microsoft.com/en-us/download/details.aspx?id=29884.
Migues, S., Steven, J. and Ware, M. (2019) *BSIMM10 LICENSE*.
Mohammad, A., Alqatawna, J. and Abushariah, M. (2017) 'Secure software engineering: Evaluation of emerging trends', in *ICIT 2017 - 8th International Conference on Information Technology, Proceedings*. Institute of Electrical and Electronics Engineers Inc., pp. 814–818. doi: 10.1109/ICITECH.2017.8079952.
Mougouei, D. *et al.* (2013) 'S-Scrum : a Secure Methodology for Agile Development of Web Services', 3(1), pp. 15–19.
Nunez, J. C. S., Lindo, A. C. and Rodriguez, P. G. (2020) 'A preventive secure software development model for a software factory: A case study', *IEEE Access*, 8, pp. 77653–77665. doi: 10.1109/ACCESS.2020.2989113.
OWASP (2017) 'Software Assurance Maturity Model Version 1.5', (Cc), p. 72.
OWASP Dependency-Check (no date). Available at: https://owasp.org/www-project-dependency-check/ (Accessed: 23 February 2020).
OWASP Dependency-Track (no date). Available at: https://owasp.org/www-project-dependency-track/ (Accessed: 23 February 2020).
OWASP Risk Rating Methodology (no date). Available at: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology (Accessed: 4 July 2020).
OWASP ZAP (no date). Available at: https://owasp.org/www-project-zap/ (Accessed: 23 February 2020).

Pal, N. and Pantaleo, D. C. (2005) *The agile enterprise: Reinventing your organization for success in an on demand world*, *The Agile Enterprise: Reinventing your Organization for Success in an On Demand World*. Springer US. doi: 10.1007/b106720.
Rindell, K., Hyrynsalmi, S. and Leppänen, V. (2015) 'A comparison of security assurance support of agile software development methods'. doi: 10.1145/2812428.2812431.
Saini, V., Duan, Q. and Paruchuri, V. (2008) 'Threat modeling using attack trees', *Journal of Computing Sciences in Colleges*, 23(4), pp. 124–131.
Saitta, P., Larcom, B. and Eddington, M. (2005) *Trike v.1 Methodology Document [Draft]*. Available at: http://www.opensource.org/licenses/mit-license. (Accessed: 23 February 2020).
Scarfone, K. *et al.* (2008) *Special Publication 800-115 Technical Guide to Information Security Testing and Assessment Recommendations of the National Institute of Standards and Technology*.
Schütz, D. and Salecker, J. (2008) 'Bill Of Material', in. Available at: https://www.researchgate.net/publication/277270226_Bill_Of_Material (Accessed: 28 June 2020).
Schwab, K. (2019) *The Global Competitiveness Report*.
SDL-Agile: Microsoft's Approach to Security for Agile Projects | Tech·Ed Europe 2009 | Channel 9 (no date). Available at: https://channel9.msdn.com/Events/TechEd/Europe/2009/SIA205 (Accessed: 31 January 2020).
Shevchenko, N. *et al.* (2018) *THREAT MODELING: A SUMMARY OF AVAILABLE METHODS*.
Singhal, S. A. (2014) 'Selection of security activities for integration with Agile methods after combining their agility and effectiveness', *International Journal of Web Applications*, 6(2), pp. 57–67. Available at: https://pdfs.semanticscholar.org/f786/b34d6aa7d83314a7307e59e4ab683131bf6f.pdf.
Siponen, M., Baskerville, R. and Kuivalainen, T. (2014) 'Integrating Security into Agile Development Methods Philosophy of Science and Information Systems View project IT Culture View project Integrating Security into Agile Development Methods'. doi: 10.1109/HICSS.2005.329.
Static Code Analysis | OWASP (no date). Available at: https://owasp.org/www-community/controls/Static_Code_Analysis (Accessed: 28 June 2020).
Stoneburner, G., Goguen, A. and Feringa, A. (2002) *Risk Management Guide for Information Technology Systems Recommendations of the National Institute of Standards and Technology*.
STRIDE chart - Microsoft Security (no date). Available at: https://www.microsoft.com/security/blog/2007/09/11/stride-chart/ (Accessed: 23 February 2020).
Tapke, J. *et al.* (no date) *I E 361 House of Quality Steps in Understanding the House of Quality House of Quality Steps in Understanding the House of Quality*.
The CLASP Application Security Process The CLASP Application Security Process Introduction 1 (2005). Available at: https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf (Accessed: 17 February 2020).
The STRIDE Threat Model | Microsoft Docs (no date). Available at: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN (Accessed: 23 February 2020).
Threat Modeling | Microsoft Docs (no date). Available at: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10)?redirectedfrom=MSDN (Accessed: 23 February 2020).
Tian, C. *et al.* (2007) 'Business value analysis of IT services', in *Proceedings - 2007 IEEE International Conference on Services Computing, SCC 2007*, pp. 308–315. doi:

10.1109/SCC.2007.36.
Tuominen, M. (2004) ‘Channel collaboration and firm value proposition’, *International Journal of Retail & Distribution Management*, 32(4), pp. 178–189. doi: 10.1108/09590550410528953.
UML Activity Diagrams - Graphical Notation Reference (no date). Available at: https://www.uml-diagrams.org/activity-diagrams-reference.html (Accessed: 3 July 2020).
Vulnerability Scanning Tools | OWASP (no date). Available at: https://owasp.org/www-community/Vulnerability_Scanning_Tools (Accessed: 28 June 2020).
What is DevSecOps? Defined, Explained, and Explored | Forcepoint (no date). Available at: https://www.forcepoint.com/es/cyber-edu/devsecops (Accessed: 9 March 2020).
Woodall, T. (2003) ‘Conceptualising “value for the customer”: an attributional, structural and dispositional analysis’, *Academy of Marketing Science Review*.

Appendix

Appendix A

Table A 1 – Target maturity values in the initial roadmap

Practice	Initial value	Phase1	Phase2	Phase3	Phase4
Strategy & metrics	0.07	0.33	0.60	0.87	1.20
Policy & Compliance	0.10	0.20	0.35	0.85	1.10
Education & Guidance	0.60	0.70	1.35	1.60	1.60
Threat Assessment	0.63	0.83	1.00	1.25	1.67
Security Requirements	1.50	1.50	1.75	2.10	2.25
Secure Architecture	1.85	1.85	2.00	2.25	2.25
Design Review	1.00	1.35	1.35	1.75	2.00
Implementation Review	0.70	1.35	2.10	2.25	2.75
Security Testing	0.00	0.52	1.27	1.90	2.22
Issue Management	0.60	0.60	0.83	1.43	1.58
Environment Hardening	0.45	0.45	0.45	0.60	1.00
Operational Enablement	0.75	0.75	0.75	1.00	1.10

Table A 2 - The maturity values after being updated at the beginning of the second phase

Practice	Initial value	Phase1	Phase2	Phase3	Phase4
Strategy & metrics	0.07	0.33	0.60	0.87	1.20
Policy & Compliance	0.10	0.20	0.35	0.60	0.85
Education & Guidance	0.60	0.70	1.35	1.60	1.60
Threat Assessment	0.63	0.83	1.00	1.25	1.67
Security Requirements	1.50	1.50	1.75	2.10	2.25
Secure Architecture	1.85	1.85	2.00	2.25	2.25
Design Review	1.00	1.35	1.35	1.35	1.50
Implementation Review	0.70	1.35	2.00	2.00	2.50
Security Testing	0.00	0.52	1.27	1.68	2.08
Issue Management	0.60	0.60	0.83	1.43	1.58
Environment Hardening	0.45	0.45	0.45	0.60	1.00
Operational Enablement	0.75	0.75	0.75	1.00	1.10

Table A 3 - The maturity values after finishing the second phase

Practice	Initial value	Phase1	Phase2	Phase3	Phase4
Strategy & metrics	0.07	0.33	0.60	0.87	1.20
Policy & Compliance	0.10	0.20	0.35	0.60	0.85
Education & Guidance	0.60	0.70	1.35	1.60	1.60
Threat Assessment	0.63	0.83	1.00	1.25	1.67
Security Requirements	1.50	1.50	1.50	2.10	2.25
Secure Architecture	1.85	1.85	2.00	2.25	2.25
Design Review	1.00	1.35	1.35	1.35	1.50
Implementation Review	0.70	1.35	2.00	2.00	2.50
Security Testing	0.00	0.52	1.17	1.68	2.08
Issue Management	0.60	0.60	0.83	1.43	1.58
Environment Hardening	0.45	0.45	0.45	0.60	1.00
Operational Enablement	0.75	0.75	0.75	1.00	1.10

Appendix B

Governance	
Strategy & Metrics	
SM1	Is there a software security assurance program in place? Are development staff aware of future plans for the assurance program? Do the business stakeholders understand your organization's risk profile?
SM2	Are many of your applications and resources categorized by risk? Are risk ratings used to tailor the required assurance activities? Does the organization know about what's required based on risk ratings?
SM3	Is per-project data for the cost of assurance activities collected? Does your organization regularly compare your security spend with that of other organizations?
Policy & Compliance	
PC1	Do project stakeholders know their project's compliance status? Are compliance requirements specifically considered by project teams?
PC2	Does the organization utilize a set of policies and standards to control software development? Are project teams able to request an audit for compliance with policies and standards?
PC3	Are projects periodically audited to ensure a baseline of compliance with policies and standards? Does the organization systematically use audits to collect and control compliance evidence?
Education & Guidance	
EG1	Have developers been given high-level security awareness training? Does each project team understand where to find secure development best-practices and guidance?
EG2	Are those involved in the development process given role-specific security training and guidance? Are stakeholders able to pull in security coaches for use on projects?
EG3	Is security-related guidance centrally controlled and consistently distributed throughout the Are developers tested to ensure a baseline skill-set for secure development practices?

Figure B 1 - Security assessment questionnaire for Governance security function

Construction	
Threat Assessment	
TA1	Do projects in your organization consider and document likely threats? Does your organization understand and document the types of attackers it faces?
TA2	Do project teams regularly analyze functional requirements for likely abuses? Do project teams use a method of rating threats for relative comparison? Are stakeholders aware of relevant threats and ratings?
TA3	Do project teams specifically consider risk from external software? Are the majority of the protection mechanisms and controls captured and mapped back to threats?
Security Requirements	
SR1	Do project teams specify security requirements during development? Do project teams pull requirements from best practices and compliance guidance?
SR2	Do stakeholders review access control matrices for relevant projects? Do project teams specify requirements based on feedback from other security activities?
SR3	Do stakeholders review vendor agreements for security requirements? Are audits performed against the security requirements specified by project teams?
Secure Architecture	
SA1	Are project teams provided with a list of recommended third-party components? Are project teams aware of secure design principles and do they apply them consistently?
SA2	Do you advertise shared security services with guidance for project teams? Are project teams provided with prescriptive design patterns based on their application architecture?
SA3	Do project teams build software from centrally-controlled platforms and frameworks? Are project teams audited for the use of secure architecture components?

Figure B 2 Security assessment questionnaire for Construction security function

Verification	
Design Review	
DR1	Do project teams document the attack perimeter of software designs? Do project teams check software designs against known security risks?
DR2	Do project teams specifically analyze design elements for security mechanisms? Are project stakeholders aware of how to obtain a formal secure design review?
DR3	Does the secure design review process incorporate detailed data-level analysis? Does a minimum security baseline exist for secure design review results?
Implementation Review	
IR1	Do project teams have review checklists based on common security related problems? Do project teams review selected high-risk code?
IR2	Can project teams access automated code analysis tools to find security problems? Do stakeholders consistently review results from code reviews?
IR3	Do project teams utilize automation to check code against application-specific coding standards? Does a minimum security baseline exist for code review results?
Security Testing	
ST1	Do projects specify security testing based on defined security requirements? Is penetration testing performed on high risk projects prior to release? Are stakeholders aware of the security test status prior to release?
ST2	Do projects use automation to evaluate security test cases? Do projects follow a consistent process to evaluate and report on security tests to stakeholders?
ST3	Are security test cases comprehensively generated for application-specific logic? Does a minimum security baseline exist for security testing?

Figure B 3 - Security assessment questionnaire for Verification security function

Operations	
Issue Management	
IM1	Do projects have a point of contact for security issues or incidents? Does your organization have an assigned security response team? Are project teams aware of their security point(s) of contact and response team(s)?
IM2	Does the organization utilize a consistent process for incident reporting and handling? Are project stakeholders aware of relevant security disclosures related to their software projects?
IM3	Are incidents inspected for root causes to generate further recommendations? Do projects consistently collect and report data and metrics related to incidents?
Environment Hardening	
EH1	Do projects document operational environment security requirements? Do projects check for security updates to third-party software components?
EH2	Is a consistent process used to apply upgrades and patches to critical dependencies? Do projects leverage automation to check application and environment health?
EH3	Are stakeholders aware of options for additional tools to protect software while running in operations? Does a minimum security baseline exist for environment health (versioning, patching, etc)?
Operational Enablement	
OE1	Are security notes delivered with each software release? Are security-related alerts and error conditions documented on a per-project basis?
OE2	Do projects utilize a change management process that's well understood? Do project teams deliver an operational security guide with each product release?
OE3	Are project releases audited for appropriate operational security information? Is code signing routinely performed on software components using a consistent process?

Figure B 4 - Security assessment questionnaire for Operations security function

Appendix C

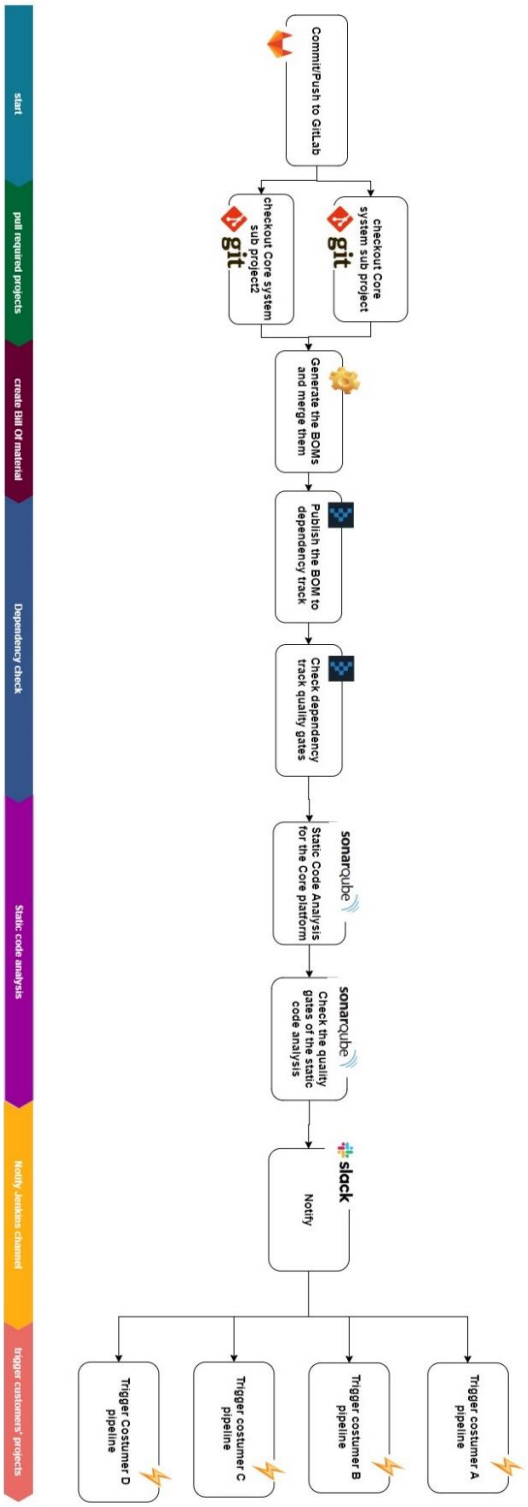

Figure C 1- CI/CD pipeline design for the Core system

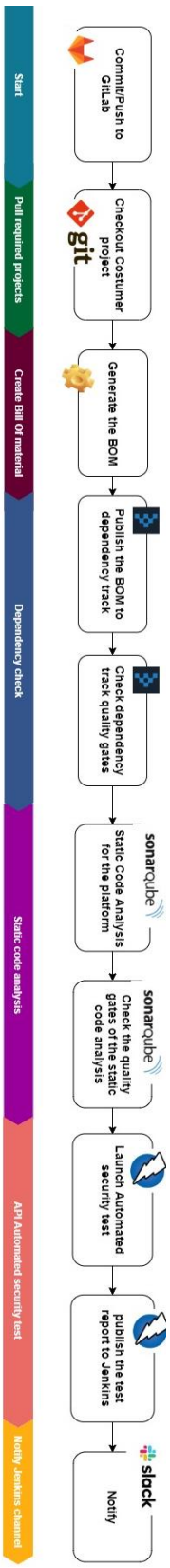

Figure C 2 - CI/CD pipeline design for Customers' project

Appendix D

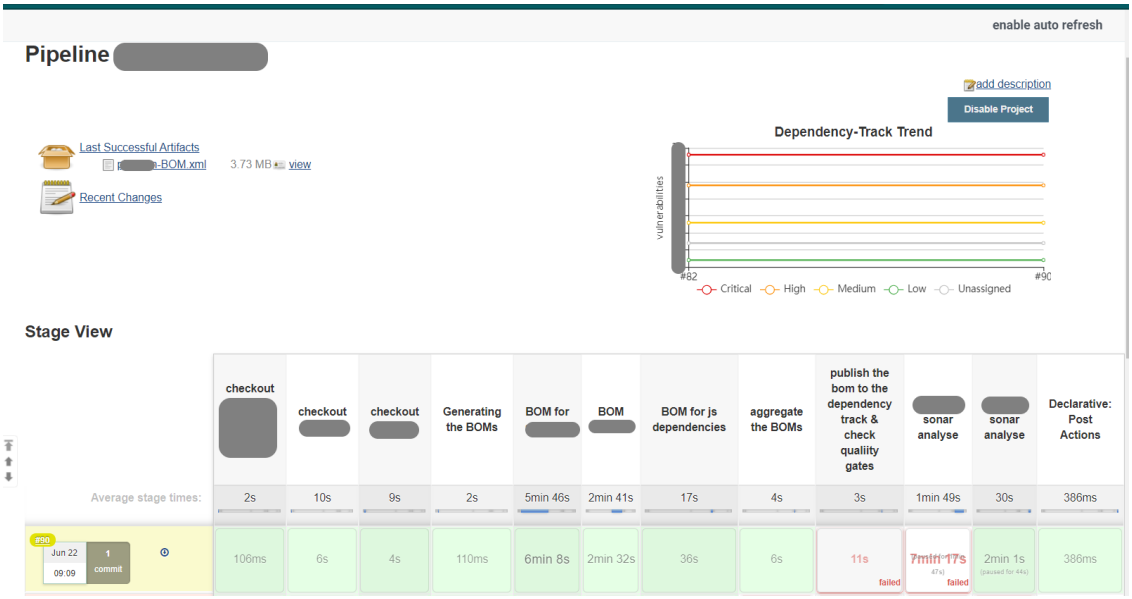

Figure D 1 - CI/CD pipeline execution for the core product in the company

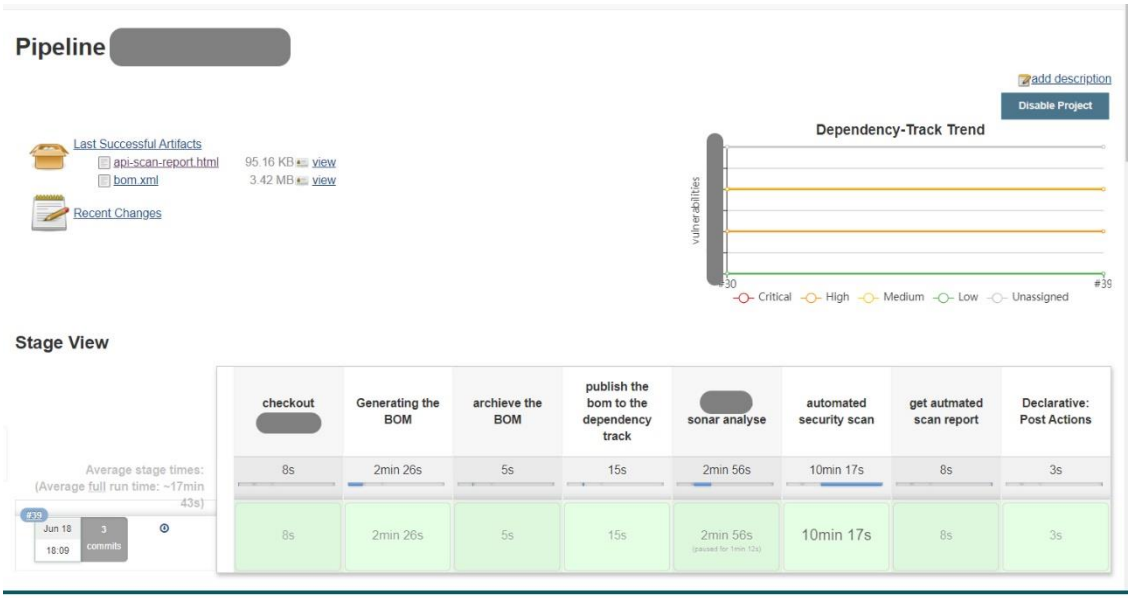

Figure D 2 - CI/CD pipeline execution for one of the customer's project

Appendix E

```
stage('automated security test'){
    steps{
        node("******"){
            checkout([$class: 'GitSCM', branches: [[name: '*/****']], doGenerateSubmoduleConfigurations: false, extensions:[[$class: 'CleanBeforeCheckout', deleteUntrackedNestedRepositories: true]], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '*****', url: '******.git']]])

            bat (script: "python ****.py", returnStatus: true)
            stash name: "api-report-scan", includes: "api-scan-report.html"
        }
    }
}
stage ('publish automated security test report'){
    steps{
        node("*****"){
            unstash "api-report-scan"
            archiveArtifacts allowEmptyArchive: true, artifacts: 'api-scan-report.html', onlyIfSuccessful: false
        }
    }
}
```

Code 3 - Integrating security test with Jenkins pipeline

```
stage('**** SonarQube Analyze'){
    steps{
        withSonarQubeEnv(installationName:***,credentialsId:'***') {
           bat'''mvn sonar:sonar Dsonar.projectKey=***Dsonar.host.url=***'''
        }
        //check the quality gate for the project
        script{
            timeout(time: 15, unit: 'MINUTES') {
                def lifeQualitygate = waitForQualityGate()
                if (lifeQualitygate.status != "OK") {
                        catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {
                            bat "Pipeline aborted due to quality gate coverage failure in : ${Qualitygate.status}"
                            bat "exit 0"
                        }
                    }
                }
        }
    }
}
```

Code 4 -Integrating SonarQube static code analysis in Jenkins Pipeline

```
stage ("publish BOM to the dependency track & check qualiity gates"){
    steps{
        dependencyTrackPublisher artifact: '*** BOM.xml',
        failedNewCritical: 1,
        failedNewHigh: 1,
        failedNewLow: 2,
        failedNewMedium: 2,
        failedTotalCritical: ***,
        failedTotalHigh: **,
        failedTotalLow: **,
        failedTotalMedium: **,
        projectId: '***',
        synchronous: true,
        unstableNewCritical: 1,
        unstableNewHigh: 1,
        unstableNewLow: 1,
        unstableNewMedium: 1,
        unstableTotalCritical: 1,
        unstableTotalHigh: 1,
        unstableTotalLow: 2,
        unstableTotalMedium: 2
    }
}
```

Code 5 - integrating Dependency Track with Jenkins Pipeline

```
stage("Generating the BOMs"){
    parallel{
        stage('BOM for project1'){
            steps{
                bat '''
                cd ***
                mvn org.cyclonedx:cyclonedx-maven-plugin:makeAggregateBom
                '''
            }
        }
        stage('BOM for project2'){
            steps{
                bat '''
                cd ***
                mvn org.cyclonedx:cyclonedx-maven-plugin:makeAggregateBom
                '''
            }
        }
        .
        .
        .
        . etc..
        stage ('BOM for javascript dependencies'){
            steps{
            bat '''
                cd ***
                npm install
               cyclonedx-bom -o js-BOM.xml
            '''
            }
        }
    }
}
```

Code 6 - Generating BOM files for Java and JavaScript in parallel

```
from xml.dom import minidom
import os
subProjectABOM = minidom.parse('../**/bom.xml')
subProjectBBOM = minidom.parse('../**/bom.xml')
jsBOM = minidom.parse('js-bom.xml')
subProjectAComponentList = subProjectABOM.getElementsByTagName('component')
subProjectBComponentList = subProjectBBOM.getElementsByTagName('component')
jsComponentList = jsBOM.getElementsByTagName('component')
# merging js bom with subProjectA bom into one temp bom
i = 0
for component in jsComponentList:
    component = jsBOM.getElementsByTagName("component")[i]
    comp = subProjectABOM.importNode(component, True)
    subProjectABOM.getElementsByTagName('components')[0].appendChild(comp)
    i += 1
bomXml = subProjectABOM.toxml().encode("UTF-8")
BOM = open("temp-BOM.xml", "wb")
BOM.write(bomXml)
BOM.close()

tempBOM = minidom.parse('temp-BOM.xml')
tempComponentList = tempBOM.getElementsByTagName('component')
# merging temp bom with subProjectB and the result is the final bom
subProjectBComponentBomRef = set()
element = 0
for component in subProjectBComponentList:
    subProjectBComponentBomRef.add(subProjectBBOM.getElementsByTagName('compon
ent')[element].getAttribute("bom-ref"))
    element += 1
for tempComponent in tempComponentList:
    if tempComponent.getAttribute("bom-
ref") not in subProjectBComponentBomRef:
        comp = subProjectBBOM.importNode(tempComponent, True)
        subProjectBBOM.getElementsByTagName('components')[0].appendChild(comp)

# fix the encoding of the resulted file
bomXml = subProjectBBOM.toxml().encode("UTF-8")
BOM = open("final-BOM.xml", "wb")
BOM.write(bomXml)
BOM.close()
os.remove("temp-BOM.xml")
BOM1 = minidom.parse('final-BOM.xml')
ComponentList = BOM1.getElementsByTagName('component')
print(len(subProjectAComponentList), "dependency from subProjectA, ", len(subP
rojectBComponentList),
      "dependency from subProjectB and", len(jsComponentList), " js were merge
d successfully into ", len(ComponentList),
      " dependency -no redundancy-")
```

Code 7 - merging two java subprojects BOMS and one JavaScript BOM into one final BOM python script