

POLITECNICO DI TORINO

Master's Degree Thesis in Computer Engineering



**Politecnico
di Torino**

Master's Degree Thesis

Definition of a DevSecOps Operating Model for software development in a large Enterprise

Supervisor

Candidate

Prof. Luca ARDITO

Valentina TORTORIELLO

Academic Year 2021/2022

Company Tutor
Francesco Floris

Abstract

Information Technology landscape is one of the fastest moving, with new products released everyday made available to millions of users: it is then very important for Technology companies to keep up with this pace if they want to be competitive. A determining factor in the success of a software development company is the adopted methodology: the trend is to switch from Waterfall and sequential methodologies, which are slow and expensive, towards Agile and iterative methodologies, that allow faster software development reducing the products' time to market. Among Agile methodologies, we will deep dive into DevOps: the aim of this strategy is to break down the siloed organization between Development and Operations teams, by composing instead cross-functional teams which have end-to-end responsibility for the product lifecycle. This is achieved through processes automation, which is key to improve the speed of development and release and also reduces human errors that are introduced in lengthy and repetitive tasks: tools are used to implement CI/CD (Continuous Integration/Continuous Delivery) of new software and functionalities, which are released more frequently and are more maintainable.

Faster development and release of software must not come at the expenses of quality and security of the product: security was often seen as an obstacle to the release of the product to the market and the consequent revenue. If in traditional sequential models the security team had the opportunity to block the release of a product as part of the "secure by design" process, in Agile models, particularly in DevOps, these security controls are more difficult to be put in place due to CD, where software and its updates are released in an almost automatic way. While Development and Operations team break down walls between them, the Security team remains isolated in its siloed structure. A DevSecOps strategy is then necessary: the aim is to make security an integrated part of DevOps processes, by considering it in each step of the SDLC. The main target of DevSecOps is not only to integrate security analysis tools in a CI/CD pipeline, but to have a real cultural shift towards a more collaborative approach to software development, mainly among Development, Operations and Security teams: the work done by one team must not be "thrown over the wall" to the other team, but every individual involved must take full responsibility on every aspect of his work, meaning that developers and operations people must be responsible of security aspects, with the support of proper professionals.

This thesis work has been carried out in the context of the Cyber Security team of Vodafone Italia, one of the main players in the telecommunications industry

worldwide, which is now aiming at become a Tech Company with a multi-year strategic program: part of this strategy includes the adoption of a DevSecOps methodology across all of the software development teams, with the insourcing of software development activities to develop products in a fast and secure way. The aim of this work has been to study the current Company setup, tools and processes and identify potential area of improvement to achieve Company DevSecOps targets. The main achievement has been the definition of a DevSecOps Operating Model made of people, processes and technologies which identify roles, responsibilities and interactions among all of the involved entities with the final goal of improving products security.

Acknowledgements

I would like to thank Vodafone for giving me the opportunity to work on this project, in particular my company tutor Francesco Floris who has followed and supported me in this journey and believed in me since its beginning. Many thanks also to the amazing Cyber Security team of Vodafone Italia, which welcomed me in their big family in the best possible way.

I would like to thank my supervisor, prof. Luca Ardito, for giving me the opportunity of writing this Master's Degree thesis.

I would like to thank my parents, Maura and Giuseppe, who have never stopped believing in me and have always been by my side, despite everything. Thanks also to my brother and sisters, Matteo, Giorgia and Martina, who supported me even in those times in which it was me who was supposed to support them.

I would like to thank Niccolò, my other half, the person with whom I spent these years at Politecnico, the ones before them and, hopefully, the ones after them: words cannot describe how thankful I am to have you by my side every day of my life.

I would like to thank all of my friends, in particular Gianluca, Erika, Matteo, Chiara and Alessandro, the ones that made those days and nights between one study session and another a bit less boring. Thanks also to anyone I met during this journey at Politecnico di Torino: every single one of you has made me who I am today.

Last but not least, I would like to thank myself, for making it to this point, for never giving up and for always having the strength to face what was put in front of you and what the future will offer you.

Contents

List of Tables	VII
List of Figures	VIII
Acronyms	X
1 Introduction to thesis context and target	1
1.1 Motivation	1
1.2 Thesis context	3
1.3 Thesis target	4
2 Introduction to DevOps and DevSecOps	7
2.1 ALM and SDLC	7
2.1.1 ALM 1.0 vs ALM 2.0	9
2.1.2 ALM vs SDLC	11
2.1.3 Advantages of integrated ALM approach	12
2.2 DevOps	13
2.2.1 Continuous Integration	14
2.2.2 Continuous Deployment	15
2.2.3 Microservices	16
2.2.4 Infrastructure as Code	17
2.2.5 Monitoring and logging	19
2.2.6 DevOps phases and CI/CD Pipelines	20
2.3 DevSecOps	22
2.3.1 Shift-left	24
2.3.2 People, processes and technology	26
2.3.3 Application Security Testing (AST)	27
3 Analysis and approach	30
3.1 Current state	30
3.2 Desired state	32

3.2.1	DevSecOps Continuum	37
3.2.2	Operating Model	41
4	Tools, resources and processes	47
4.1	Types of security analyses	47
4.1.1	Threat Modelling	48
4.1.2	SAST	48
4.1.3	DAST	49
4.1.4	IAST	50
4.1.5	SCA	50
4.1.6	VA and PT	50
4.1.7	RASP	52
4.2	Tools selection	52
4.2.1	Factors regarding product type	55
4.2.2	Factors regarding tool selection	60
4.2.3	Tools technical factors	62
4.3	Tools	62
4.3.1	Pipeline on-premises	63
4.3.2	Additional tools for cloud migration	67
4.3.3	Application security tools	68
4.4	Resources and teams	83
4.4.1	Security	83
4.4.2	Developers	84
4.4.3	IT Operations	84
4.4.4	DevOps	85
4.4.5	Business owners	86
4.4.6	Tribes, Squads and Security Champions	86
4.5	Processes	88
4.5.1	SDLC phases	88
4.5.2	SPDA phases	89
5	DevSecOps Operating Model	92
5.1	Overview	92
5.2	Plan	92
5.3	Code	94
5.4	Build	96
5.5	Test	98
5.6	Release and Deploy	99
5.7	Monitor and Respond	100
5.8	Improve	102
5.9	Decommissioning	103

6	Conclusions	105
6.1	General conclusions	105
6.2	Future developments	106
	Bibliography	109

List of Tables

4.1	Reliability ratings	71
4.2	Security ratings	72
4.3	Security Review ratings	72
4.4	Maintainability ratings	72
4.5	CVSS v2.0 classification	79
4.6	CVSS v3.0 classification	80

List of Figures

2.1	ALM components timeline	9
2.2	ALM sub-tasks	10
2.3	Comparison between ALM 1.0 vs ALM 2.0 approaches	11
2.4	Teams involved in the DevOps methodology.	14
2.5	DevOps phases.	21
2.6	Representation of shift-left concept.	25
2.7	DevSecOps concept.	25
3.1	Relative costs of fixing defects.	32
3.2	Different types of services and what they manage.	33
3.3	Major cloud providers and their 2021 and 2020 revenues compared.	35
3.4	Vodafone's DevSecOps Continuum.	37
3.5	DevSecOps Continuum: Improve and Plan phases.	38
3.6	DevSecOps Continuum: Code and Build phases.	39
3.7	DevSecOps Continuum: Test and Release phases.	40
3.8	DevSecOps Continuum: Deploy, Monitor and Respond phases.	41
4.1	AST tools decision factors	54
4.2	AST decision flowchart	56
4.3	AST tools pyramid	60
4.4	SDLC schema including SonarQube	68
4.5	SonarQube web interface homepage	69
4.6	SonarQube project overview	70
4.7	SonarQube project issues	73
4.8	SonarQube Quality Profiles	75
4.9	SonarQube Quality Gates	75
4.10	SonarQube Quality Gate condition, metric selection	76
4.11	SonarQube Quality Gate condition, threshold selection	76
4.12	Mend SCA product home page	79
4.13	Mend SCA vulnerable libraries list	81
4.14	Mend SCA library vulnerabilities list	82

4.15 Mend SCA vulnerability details	82
4.16 Vodafone Tribes, Squads and Chapters model	86
4.17 Vodafone Squads examples	87

Acronyms

ACL

Access Control List

ALM

Application Lifecycle Management

API

Application Programming Interface

AST

Application Security Testing

AWS

Amazon Web Services

BOM

Bill Of Materials

CI/CD

Continuous Integration and Continuous Delivery/Deployment

CLI

Command Line Interface

CM

Continuous Monitoring

CRUD

Create, Read, Update, Delete

CVSS

Common Vulnerability Scoring System

DAST

Dynamic Application Security Testing

DSL

Domain-Specific Language

DXL

Digital eXperience Layer

EUA

Effective Usage Analysis

IaC

Infrastructure as Code

IaaS

Infrastructure-as-a-Service

IAST

Interactive Application Security Testing

IDE

Integrated Development Environment

IDS

Intrusion Detection System

IoT

Internet of Things

IPS

Intrusion Prevention System

IT

Information Technology

KPI

Key Performance Indicator

MAST

Mobile Application Security Testing

NVD

National Vulnerability Database

PaaS

Platform-as-a-Service

POM

Project Object Model

PPT

People, Processes, Technology

PT

Penetration Testing

QA

Quality Assurance

RASP

Runtime Application Self-Protection

SaaS

Software-as-a-Service

SAST

Static Application Security Testing

SBOM

Software Bill Of Materials

SCA

Software Composition Analysis

SCM

Supply Chain Management

SDLC

Software Development Life Cycle

SLA

Service Level Agreement

SPDA

Security and Privacy by Design Assurance

SPOC

Single Point Of Contact

TDD

Test-Driven Development

UAT

User Acceptance Testing

VA

Vulnerability Assessment

VCS

Version Control System

Chapter 1

Introduction to thesis context and target

In this first chapter we will have a glimpse about some general agile development concepts which underlie below all of the conducted work and the motivations for which this thesis is being developed. Then an insight about the context of Vodafone, the company in which the work is being carried, and its project will follow. Finally the final target of this work will be presented.

1.1 Motivation

In the world of software engineering, the current trend is to go towards adopting **Agile development methodologies**.

What does Agile mean in this context? Agile software development is a term under which fall all the frameworks and practices that follow the values and principles defined in the Manifesto for Agile Software Development.

The ideas behind the Agile strategy are perfectly expressed in the **Agile manifesto**:

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

It is based on these principles that several development methodologies have been defined with a particular focus on the creation of self-organizing and cross-functional teams that collaborate actively with their customers and/or end users in order to deliver a product that is both functional and compliant to the requests of the stakeholders.

This need for faster and more reliable development technologies is driven by the demands of the businesses whose requirements are changing very fast nowadays: it may be the case that a company that needs a software has some requirements which could change from month to month, so the developers need to quickly adapt to any requirement change and deliver fast to the client. That is in fact one of the principles expressed in the Agile Manifesto.

The **DevOps** methodology falls under this concept of Agile development: the term DevOps is made by the union of the **Development and Operations** words. Historically these two worlds have been separated, the development teams are driven by the needs and the requirements of the client that requests a given product, while the operations teams are focused on the management of the production systems and their reliability, availability and performance: thus, there was a significant gap between those two realities.

In a world in which what is becoming most important is to deliver fast, this approach composed by siloed and isolated teams is not feasible anymore and needs to be changed: that is the aim to DevOps, to **remove the boundaries** between Development and Operations, including also the Testing and QA (Quality Assurance) teams: this is mostly done by introducing end-to-end automation across the whole pipeline, using appropriate tools for each of the phases of the process.

Also, in order to further break down boundaries among different teams, **cross-functional teams** are composed in order to facilitate communication among the members and work better on continuous functional feature deliveries. Another important point is the increasing complexity of the required infrastructure: this is addressed by the Infrastructure as Code (IaC) approach, which allows an easy management of almost any kind of infrastructure.

When we talk about DevOps we also talk about **CI/CD**: CI/CD stands for Continuous Integration and Continuous Delivery and Deployment. **Continuous Integration** mainly concerns the developers and it employs automation in order to compile, test and merge in a shared repository every change made to the code, so that we do not have to deal anymore with heavy merging procedures at the end of the development of some features.

Continuous Delivery and Deployment are two similar concepts, but with some important differences: delivery still requires some manual intervention by the operations team, while deployment allows the modifications by the development team to reach in an automated way the production environment.

CI and CD are put together is what is known as a **continuum**, which means that the phases that make up these processes are constantly repeated, when a cycle ends, another one is immediately started, thus making a potentially never ending process.

It is in this scenario that now we are trying to integrate security in the development process, thus obtaining the **DevSecOps methodology**: the aim of DevSecOps is to deliver the most secure product possible by performing different types of security checks during the existing phases, and not performing these checks only at the end of the product development.

This is the so called “**shift left**” **approach**: if we imagine the product development timeline as a straight line going from left to right as time passes by, shifting left means performing **security checks in the earliest development stages** as soon as possible. This brings several advantages, that in the end we can sum up with a decreased cost of defect fixing, thus an economical advantage.

So we can imagine how important it is nowadays for companies to implement a strategy such as DevSecOps that allows to develop and deliver secure products to customers.

1.2 Thesis context

The context in which this thesis has been developed is the one of **Vodafone**, one of the most important telecommunication companies, both in Italy and worldwide.

In particular we are dealing with the **DXL product**: DXL stands for Digital eXperience Layer and it is a product which has been thought to provide a standard layer for accessing frontend and backend services from many different types of channels.

The fact of having a standard layer allows every entity that needs to develop a product tailored according to the needs of a given market or customer to do so without having to worry about developing a custom solution to access a set of services which are common across the whole company, and allows also the customer to have a smooth experience even when using different channels to access the data in which he/she is interested.

The DXL product comes to life in 2018 to respond to this need: in order to be easily maintainable and scalable, DXL adopts a **microservices** architecture, with a set of exposed APIs that allow access to the backend services, both new and legacy. The microservices that compose the product are deployed using **containers** and proper **orchestration** tools, so that it is possible to have autoscaling of resources, an efficient management of failures through self-healing and maximum optimization.

The development of the DXL product is currently based on an on-premise software development pipeline following the DevOps methodology, thus using classical CI/CD tools that are hosted on physical machines and hardware belonging to Vodafone itself.

The current software development pipeline is based on some of the most popular available tools, each phase has its dedicated product(s): we have products such as GitHub and BitBucket for code repositories, Jenkins for automating the build and deploy stages, Openshift for container orchestration and many more tools.

1.3 Thesis target

The target of the work performed during this thesis is to support the migration from the current on-premises software development pipeline to a **cloud-based pipeline**, relying on the AWS (Amazon Web Services) cloud. This transition gives also the possibility to introduce new tools into the pipeline: in particular we will talk about various types of security analysis tools that are being integrated in order to implement a DevSecOps strategy.

There are several possible enabling technologies for this strategy, two of the tools on which I am going to work are two tools with different purposes:

- **Mend SCA**: the SCA open source solution provided by Mend (former WhiteSource) is one of the most popular on the market. SCA stands for Software Composition Analysis and the purpose of this tool is to find any vulnerability, bug or flaw that may be present in the third-party open source libraries used for developing a product. Eventually the tool can also look for available patches and automate their update in the project affected by the vulnerability;
- **SonarQube**: this is a code review tool that statically analyzes the source code of a project in order to provide code quality reports. Recently it also implemented SAST functionalities: SAST stands for Static Application Security Testing and consists in the search for vulnerabilities that raise from

badly written source code. SonarQube provides detailed reports on the vulnerabilities present in the project, their severity and how could they possibly be fixed.

Both tools can be easily integrated in the pipelines so that the executions of their analyses are automated as the rest of the pipeline.

One of the main characteristics of the SonarQube tool is that it is possible to define some rules (called **Quality Gates**) that block a product from being released in the subsequent environments if certain requirements are not met: rules are initially defined according to the proper policies in matter of code security, and then refine and tune these rules in order to optimize the search for vulnerabilities and avoid as much as possible finding false positives, which badly impact the efficiency of the work performed by developers.

In order to effectively implement the DevSecOps strategy, what we need is to define an **operating model** that will allow to define the involved **people, processes and technologies** and their **roles and responsibilities**. We have several parties in the company involved in this process, such as the development team, the security team and the operations team. The goal is to clarify the roles of each party and define what interactions there will be among the teams and how they will be handled.

Chapter 2

Introduction to DevOps and DevSecOps

In this chapter we will introduce some of the main topics which are needed to gain a real understanding of the work carried out in this thesis, starting from the general concepts about ALM and SDLC, and then going into the details of the DevOps and DevSecOps methodologies.

2.1 ALM and SDLC

ALM stands for **Application Lifecycle Management**: this term is used to identify a set of processes and techniques used to manage everything regarding the lifecycle of an application, from the needed tools up to the people involved in the processes.

Everything regarding the existence of an application falls under the concept of ALM, starting from the initial ideas and concepts about the application and the definition of its requirements, going through its development and testing and ending with its maintenance and possibly decommissioning.

The aim of ALM is to put together and integrate all those product management methodologies and steps that were once considered one separate from the other, such as requirements management, software development, testing and quality assurance, deployment and maintenance. This merging of activities in one single workflow allows to have a better project management overall thanks to the enhanced collaboration of all the involved teams.

ALM supports agile methodologies and DevOps, thus allowing to have continuous deployment of the software, having up to several deployments in a single day instead of having releases every few months or once a year.

As Chappell reports in [1], we can identify three main components of the ALM:

- **Governance:** it is the only part that spans the whole life of the product, from the requirements definition up to the maintenance and eventual dismissal. It includes the business case development, after which the project portfolio management starts along with the development of the application. Finally when the application is deployed and enters the organization's portfolio, we enter the application portfolio management phase in which it is needed to manage all the different applications related to the company;
- **Development:** at the end of the business case definition, the development of the application starts. Once the application is deployed, the development phase does not end, it may be needed to develop new features or fixing some existing ones, thus creating possibly frequent releases. This means that inside the whole lifetime of the product, we may have several development cycles, and this results in having several SDLC inside the ALM related to one single product;
- **Operation:** once the application has been developed, it needs to be deployed and maintained, and this requires a proper management phase. The application needs to have a proper infrastructure to run in order to meet the requirements defined in the beginning phase and its correct functioning needs to be constantly monitored. It is also very important to correctly manage the deployment of the updates, so that a smooth experience is always granted to the users of the final product.

These three phases run all in parallel, as we can see in Figure 2.1, being governance the most important part that runs from the initial concept to the end of life of the product.

As we can see in the figure, there are three main steps that determine the lifecycle of the product:

- The **idea**, which marks the beginning of the product's life and its governance phase;
- The **deployment**, which marks the end of the first development phase and the start of the operations one;
- The **end of life**, which ends all of the phases regarding the given product.

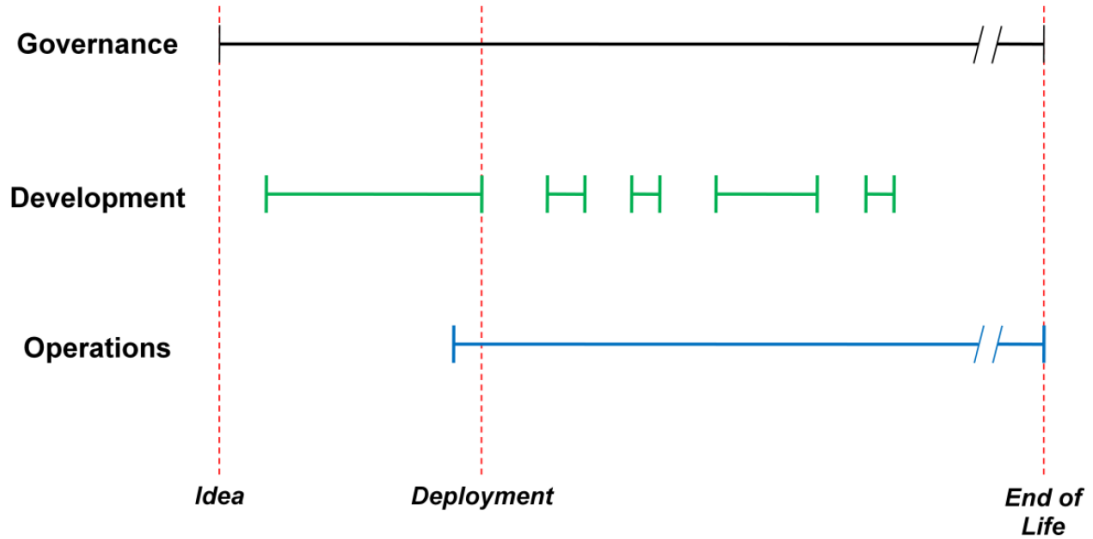


Figure 2.1: ALM components timeline

Each of the three main phases has a series of sub-tasks to be managed inside. In Figure 2.2 we can see what these sub-tasks are and the relations between the different phases.

As we previously said, the governance phase is crucial in order to manage and orchestrate the work of the other two phases, but also the development and operations phases must collaborate and coordinate themselves in order to provide the best possible experience to the final user of the product.

For example, when the development of an update has been finished by the developers team, coordination with the operations team is needed in order to ensure to have a zero-downtime deployment of the update. Since the operations team is also in charge of monitoring and collecting customers' feedback, we need collaboration with the development team to report the needed changes and develop an update to respond to those needs.

2.1.1 ALM 1.0 vs ALM 2.0

The ALM approach has changed over the years: as reported by Schwaber, cited by [2], the first proposed approach, **ALM 1.0**, lets each phase of the lifecycle to have its own management product and own data repository. This version allows to have **independence** in choosing the best tool for each of the phases and requires **less orchestration**, but on the other hand we have to face possible **synchronization issues** when integrating the artifacts produced by the different phases: another

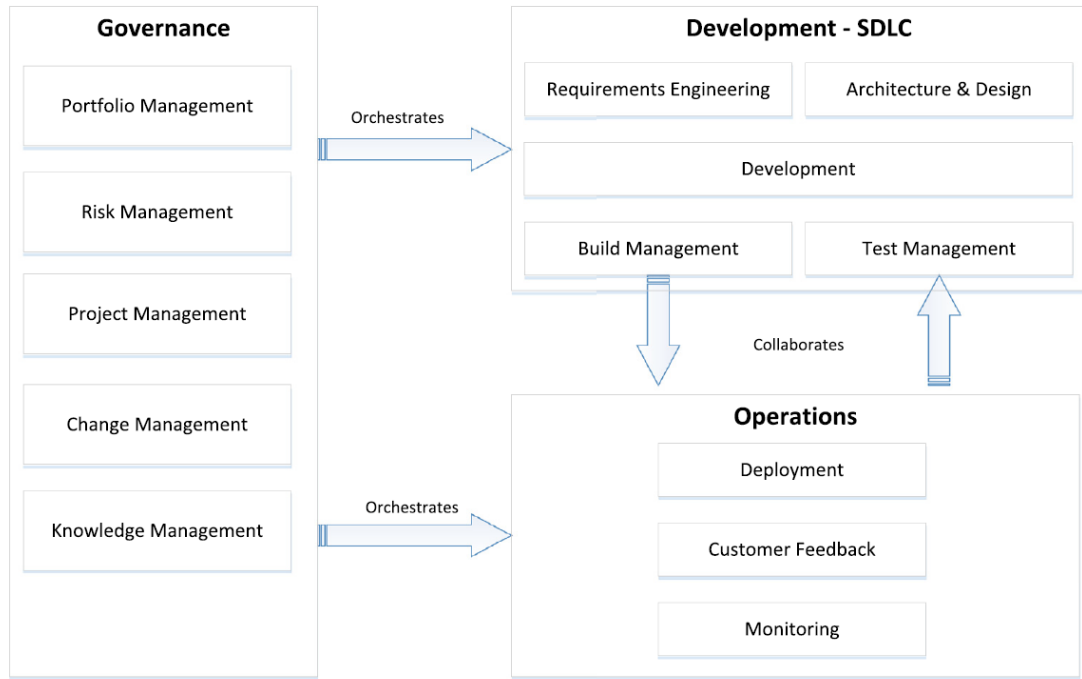


Figure 2.2: ALM sub-tasks

problem is that this integration has to be carried out mainly in a **manual way**, since we have isolated databases whose information needs to be put together in some way.

ALM 2.0 comes in order to face some of these issues by going towards a more integrated approach: instead of choosing the best-of-breed standalone product for the single phases, the main idea is to have a **single vendor** to provide the entire **suite of tools** that are needed to manage the product lifecycle, so that the integration between the artifacts of the different phases is made much simpler and can be carried in an automated way, which is easier to manage and less prone to errors. Integration is made even easier also thanks to the presence of a centralized data repository where all the steps of the lifecycle store their data and results. In Figure 2.3 we can see a graphical representation of the differences between the two approaches.

It is very important for a company which wants to create and deliver successful products that a proper set of products is chosen, being it either a collection of standalone products for each phase, or, better, a suite of integrated products that allows easy collaboration among the different teams by providing visibility to all needed information in real time to everyone that needs it. Having the right tools

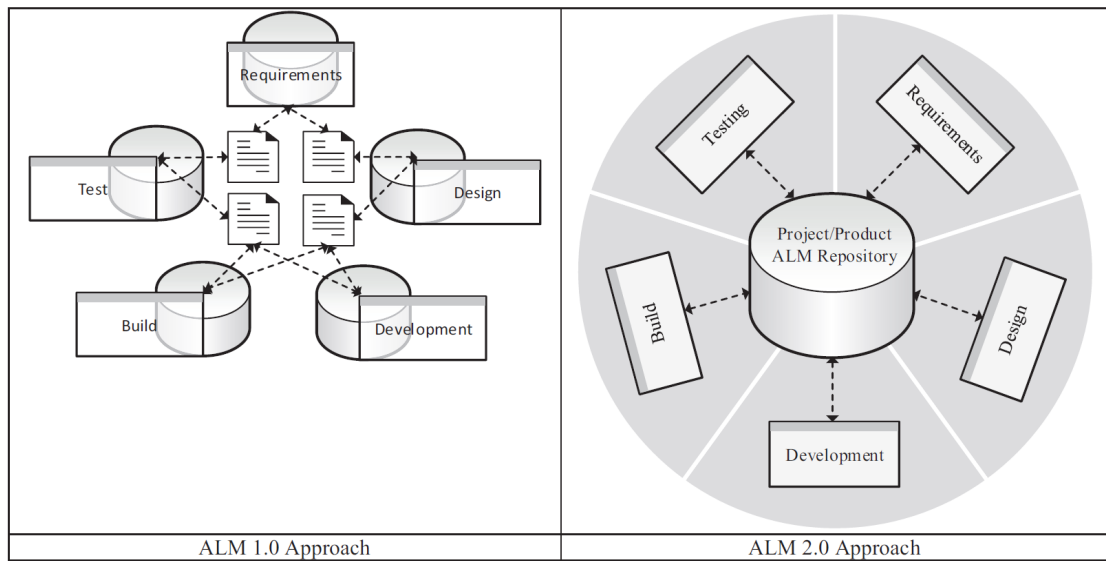


Figure 2.3: Comparison between ALM 1.0 vs ALM 2.0 approaches

to create new products to be placed in a given market can help a lot the company in terms of productivity and increased revenues.

Some examples of ALM products are:

- Atlassian Jira;
- IBM ALM solutions;
- Microsoft Azure DevOps Server;
- Tuleap (open source solution).

2.1.2 ALM vs SDLC

There is a very common misconception about the relationship between ALM and SDLC: many people wrongly think that the two terms are interchangeable and relate to the same concept, but the reality is that ALM is a broader concept and includes SDLC. In particular the lifecycle of the product, which is dealt with by ALM, spans from the very initial concept of the product up to its maintenance and eventual end of life and decommissioning. SDLC instead is a concept related just to the development part of the product and inside the ALM related to one product we may find many iterations of a SDLC, with the first one being for the initial product release and the following ones for developing and releasing new features and updates of existing ones.

2.1.3 Advantages of integrated ALM approach

Adopting an ALM approach is for sure the best choice for a software development company, but how that approach is implemented can also give significant advantages and disadvantages: the research conducted in [2] shows that an implementation that follows the ALM 1.0 methodology is not feasible for a large company in the long term, since it requires a very large effort for integrating the tools coming from different vendors and it is not a scalable solution. On the other hand it may be a suitable solution for smaller companies that cannot afford a complete suite of products of a single vendor and have smaller projects whose management is feasible with separate products.

Meanwhile, an ALM 2.0 approach is much more preferable for large companies, since the integration effort that is saved by adopting an integrated suite makes life much easier for the teams involved in the lifecycle of the products and that translates to improved productivity.

Overall, adopting an ALM strategy has some important advantages [3] which we can sum up in:

- **Improved speed, agility and real-time decision making:** using an ALM strategy helps the information flow among teams, so that each team can then make its own decisions based on updated information and then immediately share it with the other teams that can see updates in real time;
- **Improved quality and compliance:** having a standard ALM approach throughout the company makes teams to comply to the defined processes thus producing standardized products that are overall easier to manage for all the involved teams;
- **Strengthening of testing practices:** providing a common environment to all teams means that the testing team can easily provide information to the other teams about results of the conducted tests and what needs to be fixed, which may be eventually an urgent issue;
- **Improved management and planning:** the project management activities can be easily planned from these tools, where the overall available and allocated resources are visible so that proper decisions can be taken in order to efficiently plan the activities related to the lifecycle of the product.

2.2 DevOps

The ALM process provides an overview about what are the phases of an application's lifecycle, how they are related to each other and what types of tools can be used to implement it. **DevOps** enhances those definitions by providing an approach to Agile development and operation of IT applications: this is done by employing **automation** and **strong collaboration** between different teams that must work together in order to develop and **deploy high quality software frequently**.

The story of DevOps starts in 2009, when Patrick Debois comes up with this term to define a new methodology to help companies having frequent releases but still wanting to maintain high quality standards, in order to keep up with the market's demands and with the competitors.

The term DevOps stems from the union of the words Development and Operations, making clear since the beginning what its main target is: this software development methodology aims at breaking down the walls that historically exist in companies that have siloed organizations where each team performs tasks for which it is qualified and has very limited collaboration with other teams. In this context, the difficult collaboration between development and IT operations teams leads to an overall worse experience both for the final customer and for the involved employees that find themselves in the situation of managing the contrasting needs of the different teams: development teams are in fact driven by the requirements of the customers to be implemented in the product, while operations teams are more concerned about the management of availability and reliability of the IT infrastructure, as well as containing the related costs.

The adoption of such a methodology does not happen overnight: it implies an **organizational shift**, a change in people's mindsets in order to create cross-functional teams that bring together their expertise to deliver a high-quality product and experience to the final customer in the shortest time possible. As we can see in Figure 2.4 [4], DevOps requires the collaboration among the Development, Quality Assurance (QA) and IT Operations teams.

This cultural shift brings several advantages, other than the productivity improvement: the fading boundaries among the involved teams and the adoption of new tools imply that members of one team are somehow forced to acquire new skills outside of their original scope. Developers must collaborate with testing teams thus acquiring skills about Test-Driven Development (TDD) and Continuous Integration (CI), testers must ensure automation of test cases and operations people

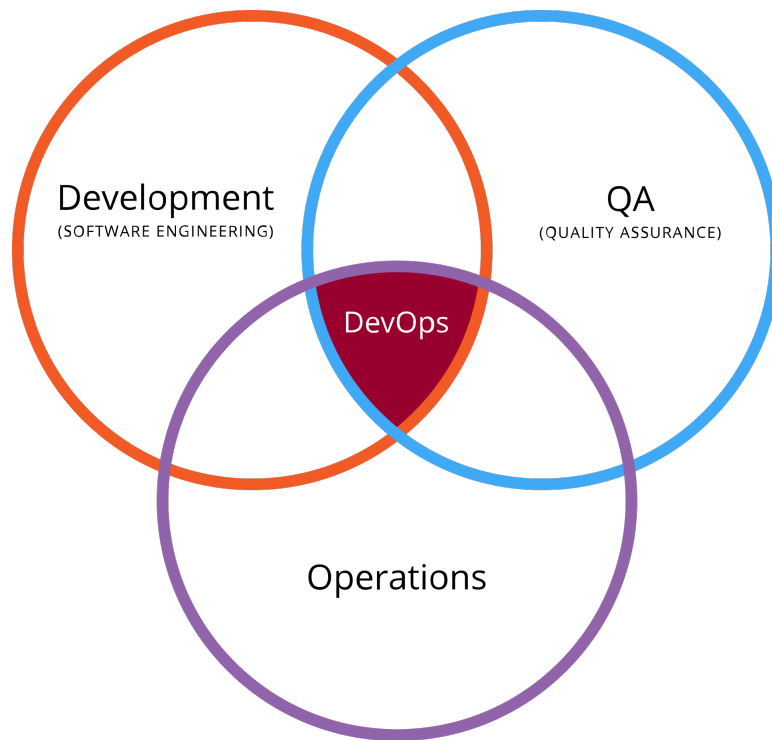


Figure 2.4: Teams involved in the DevOps methodology.

must strictly collaborate with developers by communicating about monitoring of infrastructure and application performance.

The whole DevOps methodology is based on some best practices that enable the adoption of this approach at the best of its possibilities: in the following we will present some of them.

2.2.1 Continuous Integration

Continuous Integration, better known as CI, is the practice that requires developers working on a project and writing code for it to frequently commit their changes to a common VCS repository, so that the whole team is aligned and works always on a common codebase. Also, each time some new code is committed, automated test cases are triggered and executed on the newly pushed code, in order to ensure that the application always works as expected and that the update does not break anything in it [5].

This practice fits very well in the purpose of DevOps of having shorter development cycles and more frequent releases: CI comes in to solve the problem of several developers working independently on a project and then having to integrate their works of possibly several months, making integration a not so frequent but very time-consuming task that implied many reviews and reworks and adding significant delays to the whole project.

The idea of DevOps is that these tasks that require a lot of time to be completed have to be performed more frequently: committing code once a day allows reviews to be much shorter and issues to be identified and resolved as soon as they are created. This requires the complete automation of building and testing phases, since they are time-consuming and error-prone activities if performed manually: tools that help in this exist and should be adopted, such as CI servers. Collaboration with the Testing and QA team is essential in this phase in order to be sure to implement it effectively and improve productivity.

The adoption of a proper CI practice allows performing also Continuous Delivery, since new code pushes that are successfully built and pass all the tests can also be automatically delivered to the following pre-production environments: this is the starting point for Continuous Deployment.

2.2.2 Continuous Deployment

Continuous Deployment (CD) is the continuation of the CI stage: in fact, this step of the process needs to be backed up by a solid integration phase, where code is adequately built and tested in several environments, so that as a following step that new update can be automatically deployed to the production environment without any manual intervention [6]. It is clear that this is an approach that cannot be implemented for every type of software project, but there are cases in which it allows to quickly respond to required changes, sometimes being a matter of hours between the code development and the release of the update.

Adopting a CD approach makes the deployment of updates smoother and less risky: the release of smaller updates reduces the risk of introducing some important bug or vulnerability by making easier to locate it and resolve it and the release can be deployed through a rolling update, without having to interrupt any service by temporarily taking down the servers to be updated. To further reduce risks, a canary deployment can be performed, so that the update is released to a small set of users that become testers, allowing the collection of metrics which will be used to determine if the update does introduce any failure or unexpected behaviour and

if it can be deployed to a larger set of users. Also this step can be automated by introducing thresholds on these metrics that will determine either the continuation of the deployment or the rollback of the update due to the presence of any issue. Measurement of the metrics continues also in the monitoring phase, after the update has been completely deployed, in order to be ready to respond to any bug that slipped through the review process.

CD is an approach that requires a very strong culture of collaboration, which is more important than the availability of the best of breed enabling technologies: deployment is a step that involves many stakeholders, not only developers, but also business owners, security team, and so on. It is then really important that they are continuously informed so that proper feedback can be given out at any time, this can be done through functionalities provided by CI servers such as dashboards and notifications which are always updated in real-time.

2.2.3 Microservices

As [7] reports, **microservices** are an architectural pattern that structures a software application as a collection of services with given characteristics:

- Highly maintainable and testable;
- Loosely coupled;
- Independently deployable;
- Organized around business capabilities;
- Owned by a small team.

The microservices architecture enables the rapid, frequent and reliable delivery of large and complex applications, thus being a perfect fit for the DevOps methodology.

Each one of the services that compose an application implements a function that can be independently developed and deployed in a distributed environment, where services can communicate through lightweight messaging protocols in order to implement the functionalities of the application. Due to its nature, the microservices architecture fits well into cloud-native applications, serverless computing and containerized deployment.

Microservices bring several advantages compared to the monolithic architecture, in which applications are structured into layers and then packaged and delivered as a whole, particularly in the context of a DevOps process [8]:

- **Modularity:** being the application composed by several independent smaller modules, for each of them it is easier to develop, test and maintain it;
- **Scalability:** each microservice runs as a separate and independent process, so it is possible to deploy and scale them depending on the current needs of the system in terms of traffic and needed resources, and each service can be individually monitored and properly configured;
- **Distributed development:** development of the services can be easily parallelized by making each team having end-to-end responsibility for one single service;
- **New technology adoption:** in smaller independent services it is easier to adopt new technologies to implement the required functionalities, since there are no particular dependencies to be updated, while in a monolithic application there could be a significant impact in terms of required changes in modules when upgrading or changing any technology used in the application development.

There are also some drawbacks and things to be carefully taken into consideration when choosing to adopt a microservices-based architecture for an application:

- **Testing:** end-to-end testing of application functionalities is more difficult in a microservices-based application where a single functionality to be tested could be based on several different services that need to be active at the same time;
- **Distributed systems:** microservices-based application bring along the complexity that is intrinsic to the management of distributed systems, including the management of multiple databases and their consistency;
- **Deployment:** while microservices allow a more granular scalability in terms of deployed resources for each functionality, this requires a more sophisticated approach for configuring, deploying, scaling and monitoring such services, because each of them could possibly have a different number of runtime instances to be managed at the same time.

2.2.4 Infrastructure as Code

Infrastructure as Code (IaC) is an approach to IT infrastructure management and provisioning that allows shifting from the manual configuration of physical machines to an automated process based on configuration files written in a proper machine-readable language. Both physical and virtual machines can be configured through these files, which are usually stored in a VCS, exactly as any other artifact

related to an application.

There are two main approaches to IaC:

- **Declarative:** also known as functional approach, in this case configuration files describe what are the needed resources and what is the desired state of the system, then the tool will take care about triggering the needed actions to reach such final configuration;
- **Imperative:** also known as procedural approach, in this case configuration files state how the desired configuration has to be obtained by defining the specific commands to be executed in sequence to get to the final configuration.

The IaC approach brings several benefits [9]:

- **Cost reduction:** manual hardware configuration is a very expensive task in terms of time, thanks to IaC we can configure a single machine as well as a large number of machines, both physical and virtual, just with few lines of code. This allows moving saved efforts on other important tasks;
- **Speed:** the time reduction in the provisioning of the infrastructure means that also applications can be deployed in a shorter time, improving speed of the overall development process;
- **Risk reduction:** IaC configuration files are tracked into VCS exactly as other project files and artifacts, thus it is easier to detect changes and possible related risks;
- **Consistency:** IaC helps in having a consistent configuration over all of the involved machines and environments, with no ad-hoc configurations which are not repeatable and not documented, automation of this process reduces errors due to manual configuration;
- **Efficiency:** having the infrastructure represented by means of a standard file containing its configuration allows having a predefined template that can be used to easily replicate the desired infrastructure configuration in another environment, without having to perform lengthy manual configurations.

All of these are very important characteristics in an enterprise DevOps context, where efficiency and speed brought by automation are key elements of the software delivery process. IaC plays also a fundamental role in the collaboration between developers and operations teams: the former become more involved in the definition of the infrastructure configuration, while the latter save a lot of time in case of configuration changes. Having standard IaC files facilitates the collaboration between the teams by having a common and always updated description about the needed infrastructure.

2.2.5 Monitoring and logging

Once the developed software goes into production, it is important to monitor it. In order to identify any kind of issue in its operation: this is done through **Continuous Monitoring** (CM), an automated process that exploits data and analytics gathered during the operating phase of a system in order to identify issues, inconsistencies, security breaches, etc., communicate them to the interested parties and finally resolve them. CM is also used to analyse the product's performance by tracking users' behaviours and feedback and use them to continuously improve also in terms of business. Logging any kind of event happening in the system allows the teams to perform a subsequent analysis of unexpected behaviours and understand what can be done in the future to prevent such problems.

There are several types of Continuous Monitoring, depending on what aspect of the product is being monitored [10]:

- **Infrastructure monitoring:** this includes monitoring of availability, performance, efficiency, reliability and security of the infrastructure upon which the system is running. For these purposes metrics about disks, memory and CPU usages of machines are collected and if any anomaly arises it must be properly managed to avoid any loss of quality of the provided service;
- **Application monitoring:** this includes monitoring of the application's traffic load and volume and how it is managed. The gathered data are then visualized in the form of reports and dashboards that report the results of the analyses, based on which decisions are taken. This monitoring is very important in order to ensure that the agreed SLA requirements are met;
- **Network monitoring:** this includes analysis of performance of network devices (routers, switches, firewalls, etc.) and their optimization.

Continuous Monitoring brings several advantages in the software lifecycle management:

- **Improved security:** continuous collection of performance metrics allows automation of different kinds of security measures and real-time reporting and feedback allows quick responses to security breaches by the proper team;
- **Improved performance:** CM tools can potentially be introduced in environments prior to the production one, thus allowing to identify potential problems in the system operation and prevent them prior to deploying the application in production;
- **System downtime reduction:** CM allows to quickly detect anomalies in any part of the system infrastructure and respond to them before they cause

downtimes in the services provided by the system, which, depending on its purpose, may cause also a loss in the business revenues;

- **Improved business performance:** avoidance of system downtimes and monitoring infrastructure's health finally improves the user experience and the overall business' credibility.

2.2.6 DevOps phases and CI/CD Pipelines

The DevOps process is often depicted through a representation that recalls the infinity symbol, with its left side including the Dev part of the process, while the right side includes the Ops stages: even if we can distinguish the two sides of the overall process, it must be noted that it is also well represented the fact that these two parts are strictly related and one always follows the other one, in a potentially infinite cycle, as we can see in Figure 2.5. Also, there is no net boundary between one stage and the following one, since the whole process should be followed by a single cross-functional team that includes different figures and takes end-to-end responsibility for the whole product [11].

We will now see some details about each phase of this process.

Plan

This is the phase that takes place before developers start writing code, in which the Product Owner and the involved stakeholders express their requirements: these are then formalized into a product backlog which is used as a base to plan the activities for the sprints, where tasks are allocated to the developers for them to work on a particular feature.

Code

In this phase developers work on the assigned tasks using their tools, such as IDEs integrated with proper plugins to support and make this step more efficient by writing good quality code since the beginning of its development.

Build

The Build phase is the one in which the DevOps approach really starts to make a difference with respect to other approaches: at the end of the Code stage, developers commit their work to the common repository that is shared with the whole team by opening a pull request to be reviewed. By employing CI tools, this step triggers an automated build process including the execution of some unit, integration and end-to-end tests: if any of the build or test steps fails, the pull request fails and the developer is notified with the issue to be solved before opening another one.

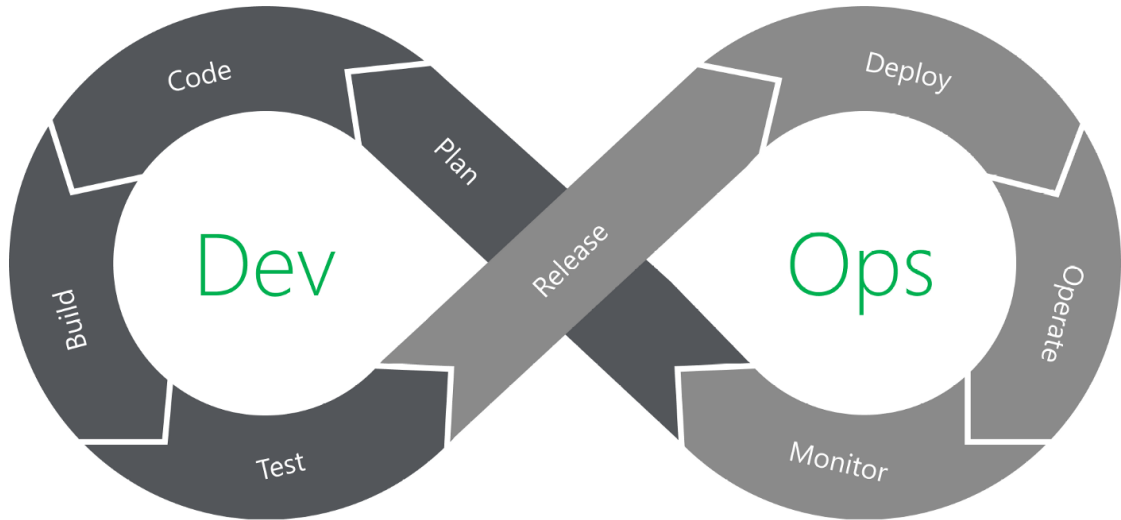


Figure 2.5: DevOps phases.

Test

After the build is completed successfully, it is automatically deployed into a testing environment, where a series of manual and automatic tests are performed: the first include User Acceptance Testing (UAT), where potential customers test the application in order to point out possible corrections to be performed before deploying it in production. Automated tests can include instead security testing and load testing.

Release

This is the crucial step in which a system that passed the build and test phases can finally be planned to be deployed in the production environment: in an ideal DevOps process, also this step would be automatically performed by using CD tools and IaC, making it possible to have up to several releases per day. If a company is not ready to embrace total automation of this process, this step can still be subject to a manual review and approval of the systems to be promoted into production.

Deploy

When the moment comes for the approved build to be deployed, the proper environment is set up and configured for the release to be available for users and customers. It is important to ensure a deployment that affects as least as possible the availability of the provided service, if for example we are deploying an update of an existing application and there are techniques to do so, such as blue-green deployment.

Operate

At this moment the system is up and providing its services to the customers, during this phase the operations team makes sure that the infrastructure is properly provisioned and adequately supports the application. Feedback from the customers must also be collected to understand how the product can be further improved in the following stages.

Monitor

In this final stage, the collected feedback from customers must be gathered in the form of data and providing analytics about the performance, errors, users satisfaction, and so on. The results of these analyses must be fed back to the Product Owner and the development team who will then start working on the new features and updates by reiterating the DevOps process.

CI/CD Pipelines

All of these stages are put together in practice in what we call CI/CD Pipelines: these pipelines are composed by a set of tools, each of them responsible for one or more of the phases explained above, and their main feature is of course the strong usage of automation, that makes the process faster and less error-prone. There are very popular tools, such as Jenkins, that allow defining such pipelines: in more recent times, cloud-based pipelines are also becoming very used, due to their ease of infrastructure provisioning, their use of containers and in general the fact that many complex management factors are outsourced to companies that can provide the required services in a fast and secure way.

2.3 DevSecOps

While the DevOps methodology aims at speeding up the SDLC still maintaining high quality standards, another critical aspect of software products to be always considered is their security: nowadays security incidents in IT systems can cause

very significant problems, mainly in terms of sensitive data losses and leakages and unavailability of provided services, which, depending on the context of application of the system, could also imply financial penalties or even legal issues. In any case, such events cause problems to the company in terms of economic revenues and reputation.

The aim of the DevSecOps approach, as the name suggests, is to **integrate security in the whole software lifecycle**: security reviews and audits are always seen by developers as an obstacle to fast development and releasing of new features and updates, but the risk of ignoring such problems is that eventual security flaws that are introduced and not timely resolved can then be exploited, eventually causing a much more significant loss in terms both of time and economical costs.

In a way very similar to the Agile methodology, a **DevSecOps Manifesto** has been written by Larry Maccherone [12] to define the principles of this new strategy: the Manifesto defines a series of contrasting aspects and the need to prioritize one way of dealing with them with respect to another. The main points are:

- **Build security in more than bolt it on**: security must be thought and implemented along with the product (security by design), the same care taken into product functionalities has to be put into security ones;
- **Rely on empowered development teams more than security specialists**: if responsibility for security implementation is shifted towards developers, they will start making different choices from the start, giving more importance to security since earlier stages;
- **Implement features securely more than security features**: security must not be considered only in those features that are strictly security-related (encryption, authentication, etc.), but all of the features and functionalities of a product must be implemented in a secure way;
- **Use tools as feedback for learning more than end-of-phase stage gates**: security tools are anything but infallible and we must not rely on them only as gates over which the product cannot pass if it does not meet the given criteria, but mainly as tools from which the developers can learn about their mistakes and not repeating them anymore;
- **Build on culture change more than policy enforcing**: usually enforcing strict security policies goes in contrast with the main target of the developers which is to deliver functionalities in a fast way, until recent times the security teams had the power in their hands by not allowing unsafe releases to go into production, but with the advent of the DevOps approach this possibility

faded away. What is more important now is to make developers understand the impact of eventual security vulnerabilities and the importance to avoid introducing them since the beginning of the SDLC.

Adopting a DevSecOps strategy clearly implies several advantages, given by the fact that we are integrating security in an Agile development strategy [13]:

- Reducing expenses due to security issues and increase of delivery rates;
- Security, monitoring, deployment checks from the beginning, with automated notifying systems;
- Supporting transparency among teams since the start of the development;
- Implementing the “Secure by Design” approach;
- Faster recovery in case of security incidents;
- Improvement of overall security by integrating automated controls in the development pipeline.

2.3.1 Shift-left

One of the main principles on which the DevSecOps methodology is based on is the **shift-left principle**: this concept has been first applied in the testing scope, where applying tests only at the end of the development phase often resulted in a bottleneck for the whole process, causing significant costs and delays for the product release.

In DevSecOps this idea is applied to the security scope: security of products must not be considered and assessed only at the end of the process, instead it must be considered since the starting phases by adopting proper tools and measures. Apart from the above-mentioned delays, not considering security in every aspect of the developed product may lead to unsecure releases containing vulnerabilities that could be exploited by malicious users, causing significant problems to the company, which must spend time and resources to manage the incident, may have reputational damages and loss of trust from customers and even face legal problems.

The concept of shift-left can be visualized by considering the timeline of the SDLC process: in this context, shifting left a phase means moving it earlier in the timeline, as we can see in Figure 2.6 from [14].

However, we must note that shifting left does not just mean that we are taking a phase that was at the end of the process and translating it to the beginning: shifting left in testing means that security must be embedded in each phase and



Figure 2.6: Representation of shift-left concept.

considered during the development of the artifacts, being it code or documents. At the end of each phase the artifacts produced must be evaluated and checked if they meet some predefined quality requirements.

The shift-left approach in security is based on a very similar idea, in fact the aim is to integrate security and related analyses and assessments in all of the phases of the DevSecOps process.

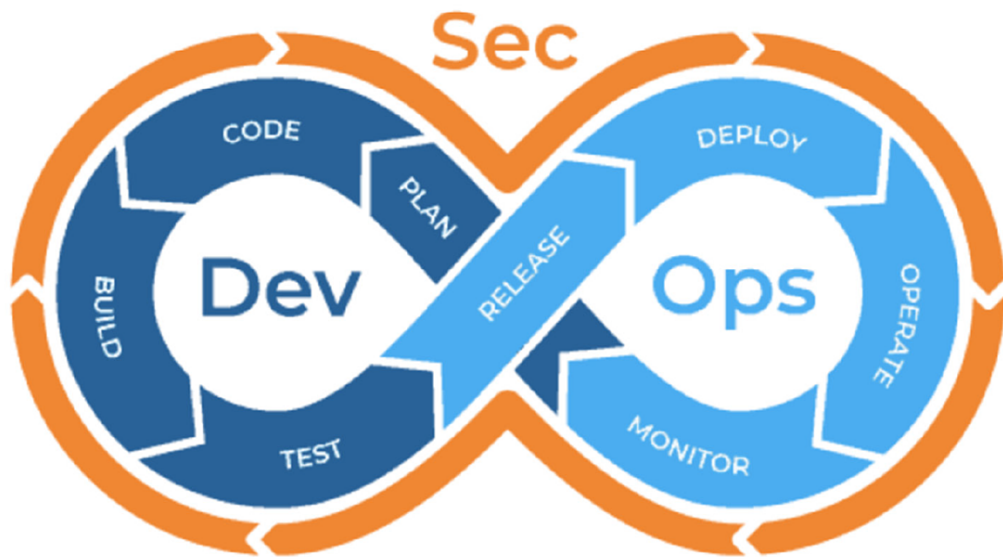


Figure 2.7: DevSecOps concept.

As we can see in Figure 2.7 the aim is to maintain the structure and the efficiency of the DevOps approach, with its phases and their sequence and integrate security in all of them: in this sense automation must be a key feature of this process, there

is no space for the manual and lengthy security reviews and audits in this process, every analysis must be automated through proper tools in order to maintain the speed and efficiency of the Agile software development process and to avoid errors which are likely introduced by humans.

The target of this approach is not only achieved by adopting the proper technical tools, but by implementing a real cultural shift in the company, how its software development teams are organized and the roles and duties of every component of the team. Following this idea, we can distinguish two main factors that are needed to effectively implement a DevSecOps strategy:

- **People, processes and technology;**
- **Application Security Testing** techniques.

In the following we will introduce them.

2.3.2 People, processes and technology

People, Processes, Technology (PPT) is a well-established framework for supporting organizational changes and transformations: it relies on the three main pillars upon which companies are based and it aims at finding the optimal balance among them, so that value and products are shipped in the best and fastest way possible. We can think at the components as the legs of a three-legged table: if one leg has a different length from the others, the table will lose balance and not perform its task correctly. In a very similar way, if one component of the PPT framework is unbalanced, then it will be more difficult to perform any task: for example, if new generation technology is adopted for supporting the processes, but people are not adequately trained to use it, it will just be a waste of time and money for the company.

The three pillars of the PPT framework are [15]:

- **People:** these are the human resources available to the company, those that perform the tasks defined in the processes, possibly leveraging on the available resources;
- **Processes:** they are the steps and actions to be performed in order to reach a goal, they define how to achieve the final desired result and how people and technology will interact to meet the business requirements;
- **Technology:** this includes all the tools and techniques that people have to use to put the processes in place. Technology is the last step to be considered in this framework, differently from what could be commonly thought: first of

all the required processes must be defined and people to implement them are to be hired, finally the proper technologies to support the process and that can be used by the hired employees are to be acquired by the company.

The PPT framework will be further analyzed in Chapter 3.

2.3.3 Application Security Testing (AST)

Application Security Testing is one of the main processes to be introduced in DevSecOps, it is divided in many different steps, each of them targeting at a different phase of the SDLC and at the testing of different types of artifacts that are produced along the product lifecycle. Automation covers a key role in this process, since it is one that requires continuous testing against the security threats that arise on a daily basis in the IT world, thus requiring companies providing products and services in this field to be constantly updated.

To be compliant with the shift-left idea, the AST process has to be integrated since the starting phases of the SDLC, through the techniques that mostly fit the particular stage taken into consideration. Furthermore, it is very important that all of the components of the software product are security tested, not only the ones that are exposed to the external world, such as APIs: if attackers succeed in getting through the external perimeter, the company's systems will be in serious danger if not properly protected.

The main techniques that are part of AST are:

- **Static Application Security Testing (SAST);**
- **Dynamic Application Security Testing (DAST);**
- **Software Composition Analysis (SCA);**
- **Interactive Application Security Testing (IAST);**
- **Mobile Application Security Testing (MAST);**
- **Runtime Application Self-Protection (RASP).**

There are other techniques which are not properly included in the AST ones, but are still very important to assess the security of a system:

- **Threat Modelling;**
- **Vulnerability Assessment (VA);**

- **Penetration Testing (PT).**

In the event that not all of the possible security measures can be applied, the characteristics of the product must be analyzed and the most suitable tools must be employed to ensure the highest possible security. Most of times it is not even necessary to apply every single tool, given the features of the product to be security tested: applying all of the possible security analyses even when some of them are not useful can result only in a waste of resources and a loss of performance and efficiency, key features of the DevSecOps methodology. These techniques and the factors to be considered in order to apply them will be further analyzed in Chapter 4.

Chapter 3

Analysis and approach

In this chapter we will analyse the current state of DevOps and its implementation compared to the desired state, that is the complete transition to a DevSecOps methodology for software development with the proper tools, resources and processes. We will also define what an operating model is, why it is needed and how it can be built and implemented effectively in an information security context.

3.1 Current state

The current Vodafone SDLC approach is based on an **on-premises infrastructure** to implement the **DevOps pipelines**: this has several advantages and disadvantages.

Having an on-premises infrastructure means that all hardware and software needed to provide some functionality or service are **completely hosted inside of the company facilities**: this allows to have full control over the machines that are deployed and how they are configured.

The main reasons why an organization would like to have an on-prem infrastructure are **security and privacy**: if we are offering services that deal with sensitive data, for example in the healthcare or military fields, there are regulations that forbid these data from being transmitted outside of the borders of a given country or region. In general, relying on a third-party service would imply sharing data with the service provider and this increases the risk of a data breach.

Other reasons for which this approach could be chosen are **customization and performance**: having full control on what hardware is selected and how it is deployed could be a determining factor for the performance of the infrastructure.

For example, having a machine that performs a given critical task to be as close as possible to the company premises could significantly improve the speed at which that service is provided and thus the customers' satisfaction.

The main disadvantages of on-premises infrastructures are their **need for maintenance** and **capability of scaling**: first, to properly configure the needed resources, they need to be requested, bought, shipped and set up by dedicated IT professionals, which overall is a process which is expensive in terms of human resources' effort, time and money. All of this must be repeated in case an upgrade of the infrastructure is needed and more hardware need to be added for scaling in order to support a higher request for services: this makes the adjustment, both in terms of extending and reducing resources, as difficult as the initial setup.

From the **application security** point of view, the state of the art is that the required analyses are performed in a **manual way** only at the **end of the development** stage: this approach has several disadvantages, which translate most of the times in a waste of time and resources. Applying security checks only after finishing the development means that if any defect is found it may be necessary to repeat the whole process that introduced the vulnerability and the effort that was previously spent on developing that feature has been wasted. This can even mean a delay in the release of the product and an increase of the time-to-market, which can cause economical and reputational damage to the company: a study by IBM, cited by [16], reports the relative costs of fixing defects in different stages of the product lifecycle in Figure 3.1.

Here we can see that fixing a defect in the Testing phase costs 15 times more than the cost in the Design phase, that is, for example, if a defect takes one hour to be fixed after the Design stage, the same defect needs 15 hours to be fixed after the Testing phase. What also is important is that the cost of fixing a defect while the product has already been deployed to production costs 100 times more than in Design phase.

Performing the security analyses manually is another source of possible errors and thus further costs: introducing automated tools will be the solution both to avoid this kind of errors introduced by the human work and improve the efficiency of the product development process.

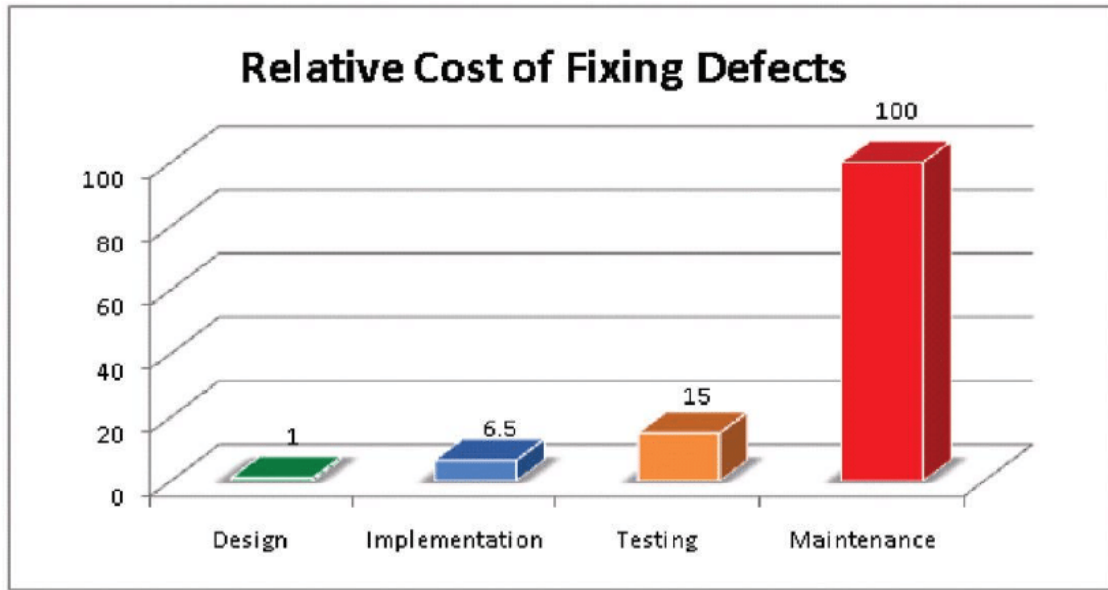


Figure 3.1: Relative costs of fixing defects.

3.2 Desired state

The target state for the Agile development model employed in Vodafone is to have a **cloud-based CI/CD pipeline** for software development following the **DevSecOps methodology**.

Cloud-based infrastructures, in opposition to on-premises ones, are hosted **outside of the company's facilities**, in particular, they are hosted and managed by a **third-party service provider**: this is the **Infrastructure-as-a-Service (IaaS)** model, which is part of the cloud computing world, and allows having access to the needed computing, storing and network resources to deploy the developed applications and services. IaaS is one of the three main models based on which cloud services are provided, the other ones are Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS): what sets them apart is what resources are managed by the company requiring the services and what is managed by the service provider, as we can see in Figure 3.2 from [17].

IaaS is the most flexible among the three types of offered services, because it offers just the basic resources upon which the company can deploy various kind of products, thanks also to the virtualization features.

There are several factors to be taken into consideration while choosing a cloud

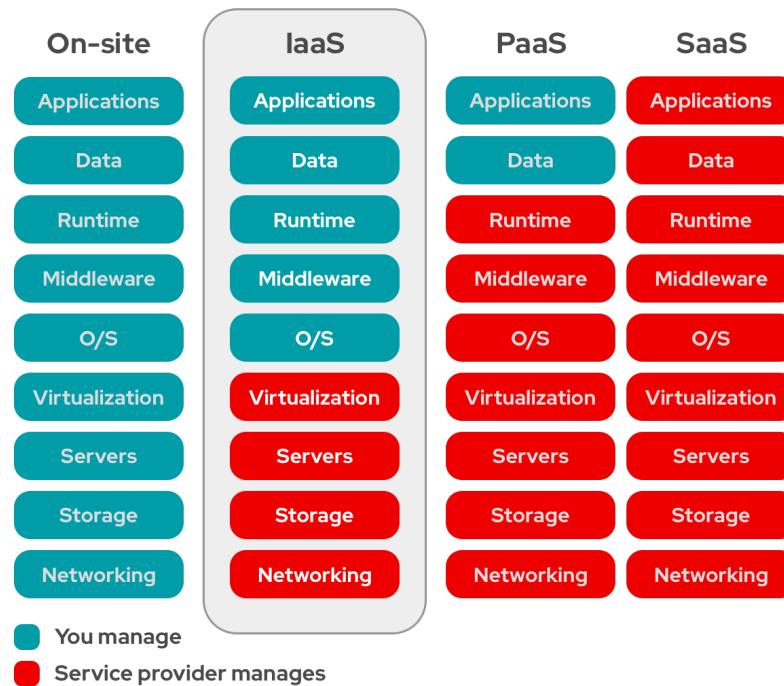


Figure 3.2: Different types of services and what they manage.

provider:

- **Flexibility:** possibility to purchase only the needed components and adding and removing them based on the current necessities of the company;
- **Reduction of costs:** not having to maintain a local data centre or any other kind of system reduces significantly the costs, because there is neither the need to buy any hardware nor to maintain it by placing it in proper rooms and hiring dedicated personnel. Most importantly, the provided services are paid based on what is effectively used, avoiding any kind of waste of economical resources;
- **Control:** how much the infrastructure can be customized and controlled based on the customers' needs and what products and services need to be deployed;
- **Security:** the provider must provide some guarantees about the security measures in order to avoid incidents and data breaches, also there must be some disaster recovery procedures in place so to ensure business continuity also in case of problems;

- **Multi-tenant systems:** the provider’s infrastructure is usually shared among different customers and it is important that one cannot access to the data of another. Also, the provider must ensure that resources are properly allocated to the requesting users and avoid situations in which a “noisy neighbour” is allocated more resources than the other users, thus slowing down the performance of the other customers using that resource;
- **Service:** it is important to be aware about the Service Level Agreement (SLA) of the service provider and what is stated about the time dedicated to the resolution of problems of resources provisioning;
- **Reliability:** the performance of the final deployed product reflects the one of the underlying infrastructure, thus it is important that the provider ensures the correct functioning and reliability of the provided hardware and software.

By taking advantage of this migration from the on-premises to the cloud-based infrastructure, the company has the opportunity to integrate **new automated security tools** in the newly implemented CI/CD pipelines: this is thanks to the flexibility of the cloud-based environment and the fact that not having to deal with the operational and security issues of the infrastructure frees some resources that the company can redirect on other tasks, in particular the security aspects of the product under development.

The integration of security into each phase of the SDLC allows the people involved in the process to find the defects in artifacts as soon as they are created and fix them before they are released to subsequent phases and environment and become source of further errors down the chain, which then result in higher cost and effort required for fixes.

This approach allows to **embed security** into the software product, making it a design constraint rather than something that is eventually later assessed: this is the “**secure by design**” strategy whose aim is to ensure that a product is secure from its foundations, by considering several security strategies since the design stage and enforcing these constraints through suitable architectural choices. The security patterns adopted in this approach are widely known and reusable and provide solutions for every required security feature, such as authentication, authorization, confidentiality, data integrity, privacy, accountability, availability, safety and non-repudiation [18]. Some basic principles to follow in order to adopt this strategy are:

- **Expecting attacks:** it must be assumed that attacks can be performed on our system and we can use an approach such as domain-driven design to better identify the possible threats and the proper countermeasures;

- **Avoiding security through obscurity:** Linus's Law states that "*given enough eyeballs, all bugs are shallow*" [19], this is a perfect summary of the fact that secrecy is often not a good long-term choice, particularly in coding. When design choices and code are made secret developers are more likely both to not find existing bugs and introduce new ones in updates' development;
- **Least privilege:** it is a very common principle in information security and it states that the lowest amount of privileges must be granted to a given resource or system that allows it to correctly perform its tasks.

The trend for now and the next years is to gradually migrate to cloud-based infrastructure: Gartner states that the IaaS market in the public cloud sector has already increased of 41,4% in 2021 with respect to 2020, with a total revenue of \$90.9 billion and the top-5 providers accountable for more than 80% of the market share, as we can see in Figure 3.3 from [20].

Company	2021 Revenue	2021 Market Share (%)	2020 Revenue	2020 Market Share (%)	2020-2021 Growth (%)
Amazon	35,380	38.9	26,201	40.8	35.0
Microsoft	19,153	21.1	12,659	19.7	51.3
Alibaba	8,679	9.5	6,117	9.5	41.9
Google	6,436	7.1	3,932	6.1	63.7
Huawei	4,190	4.6	2,681	4.2	56.3
Others	17,056	18.8	12,697	19.8	34.3
Total	90,894	100.0	64,286	100.0	41.4

Source: Gartner (June 2022)

Figure 3.3: Major cloud providers and their 2021 and 2020 revenues compared.

This transition is driven by the advantages that the cloud-based approach offers compared to the on-premises infrastructure:

- **Paying only for the effectively used resources:** this is an big advantage especially for those small and medium enterprises that do not have enough

resources to face the setup of an on-premises data centre. IaaS allows them to have a small initial investment and a periodic cost, which is always limited to the resources that are effectively used;

- **Fast provisioning and scaling:** cloud-based infrastructures are available in a very short time, since there is no need for the company to install and configure any hardware or software, that is one of the duties of the service provider. The infrastructure is available for deployment as soon as the subscription is completed and started, making provisioning of the infrastructure for an application a very easy and fast process. In a very similar way, it is very easy to scale both up and down the needed resources, based on the customers' and business' needs;
- **Maintenance:** no need for dedicated personnel to maintain the infrastructure, perform updates, managing malfunctions and decreased costs also in terms of space in the company's facilities and energy consumption;
- **Security:** this is another duty that is now up to the service provider, which are usually big companies that can afford to have dedicated security teams dealing with both the physical security of the machines and networks and the security of the information stored inside them. This is a huge advantage for those companies that cannot afford to have a dedicated IT security team.

In the meanwhile, as previously said, an on-premises infrastructure could be required in those cases in which a company deals with particularly sensitive data, for example in financial or healthcare sectors, where there are regulations for which data cannot cross national borders, making them unsuitable for storage in cloud servers, which most of the times are hosted in big data centres in western Europe or USA. Also, legacy strategic systems can rarely be adapted to the requirements of the IaaS services, making it necessary to host and run them on a local infrastructure.

A solution that can meet all of the possible requirements of a business is a **hybrid infrastructure**, that is one that is hosted part on-premises and part on the cloud provider's servers: for example, the part of products and services that are hosted locally could be the ones that are safety-critical or legacy components that are of strategic importance and cannot be neither dismissed nor migrated to a cloud-based infrastructure. On the other side, we can rely on remote cloud servers for example in the case of asynchronous computations which require high loads in terms of computing and networking resources, relying also on the ease of scaling that cloud-based environments offer.

3.2.1 DevSecOps Continuum

The Vodafone approach to DevSecOps is based on the so-called **DevSecOps Continuum**: this is a model based on the stages identified by the DevOps methodology which helps in understanding what are the phases in which security must be implemented, by baking the proper people, processes and technologies into the SDLC.

We distinguish **nine phases** which are depicted in the representation seen in Figure 3.4.



Figure 3.4: Vodafone’s DevSecOps Continuum.

The continuum phases are divided in two main blocks: first we have a **CI block**, which includes the planning, coding, building and testing phases, the ones in which the software product is actually devised and developed. Following the CI block, we have the Release stage which brings to the **CD block**, where the application is deployed and its functioning is continuously monitored, so that there can be proper responses to incidents and planning of future improvements and updates.

In the following we will see the details about each phase.

Improve and Plan

These are the phases in which the opportunities for improvement are identified and their realization is planned, the aim is to answer the question “what can we improve and how can we plan these improvements?”.

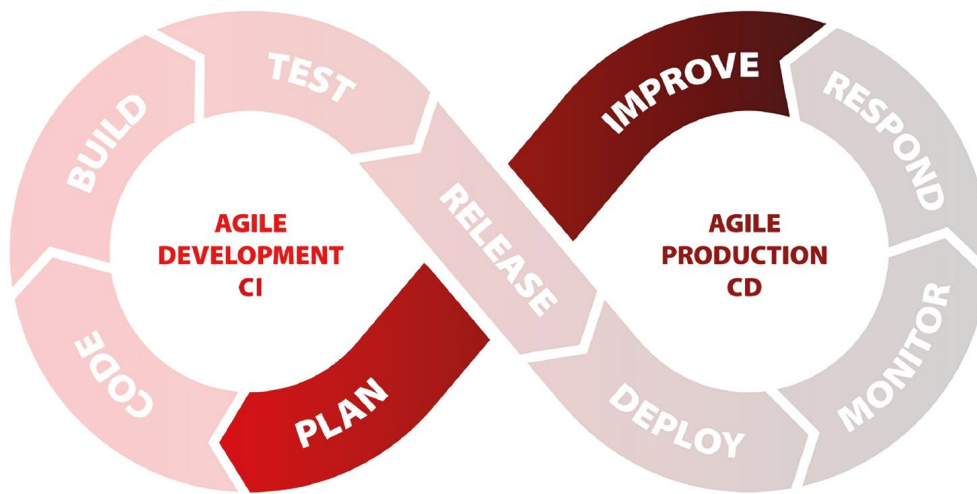


Figure 3.5: DevSecOps Continuum: Improve and Plan phases.

The main security aspects to be considered in these stages are:

- Gathering security requirements from customers and stakeholders;
- Conducting retrospective meetings to analyse what went well and what did not and what are the lessons to be learned from the security point of view;
- Including in the backlog of the product user stories that include security aspects;
- Performing threat modelling analyses on the architecture to identify the risks related to the product and decreasing the possible attack surface;
- Complying with the Secure by Design guidelines and integrate any component or control needed to mitigate security threats and enforce any required security policy.

Code and Build

These are the stages in which the real application or service is developed and built, where the defined architecture is translated in a set of components that will then work together to provide the requested functionalities.

The security aspects to consider during this phase are:

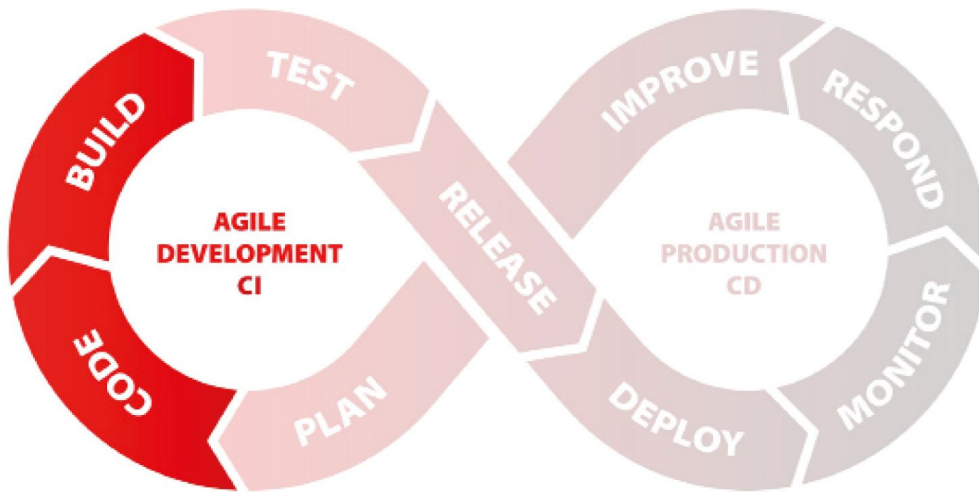


Figure 3.6: DevSecOps Continuum: Code and Build phases.

- Searching for vulnerabilities before the code is pushed into production and following secure coding best practices to reduce the introduction of flaws and vulnerabilities;
- Usage of proper SCA tools to identify vulnerabilities present in used third-party open-source libraries;
- Creation of SLAs to improve vulnerability remediation time from development teams;
- Including peer code reviews and false positives validation processes;
- Usage of security-approved tools and plugins for development;
- Considering security user stories while developing.

Test and Release

These are the phases in which a prototype or the final product are tested in order to check if they meet the requirements, both functional and non-functional. Several types of tests are performed both on the single components and on the set of them interacting (Unit and Integration tests), if the tests are successful, the release of the product is finally planned and performed.

The things to consider from the security point of view during these stages are:

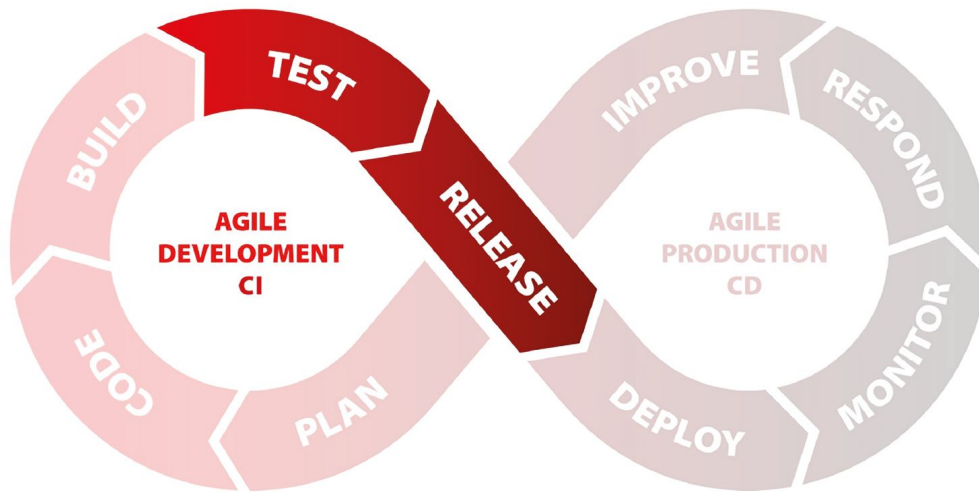


Figure 3.7: DevSecOps Continuum: Test and Release phases.

- Usage of Dynamic Application Security Testing (DAST) tools in the test and staging environments to check pushed code for flaws and vulnerabilities;
- Usage of security-approved tools to perform analyses on the cloud environments to verify configuration changes, check compliance, run performance and load tests;
- Usage of container scanning tools to detect possible misconfigurations;
- Perform penetration tests (pentests) with the support of the Secure by Design team;
- Usage of feature flags to turn on or off features that are ready or not to be released to customers.

Deploy, Monitor and Respond

These are the final phases of the cycle, the ones in which the final product or feature is deployed to the production environment and made available to the customers. The operation of the product is then monitored and checked for any suspicious activity and in such a case we must properly respond to any kind of attack.

During these final stages we must:

- Choose whether to perform manual or automatic deployment and checking of the presence of the proper security controls;

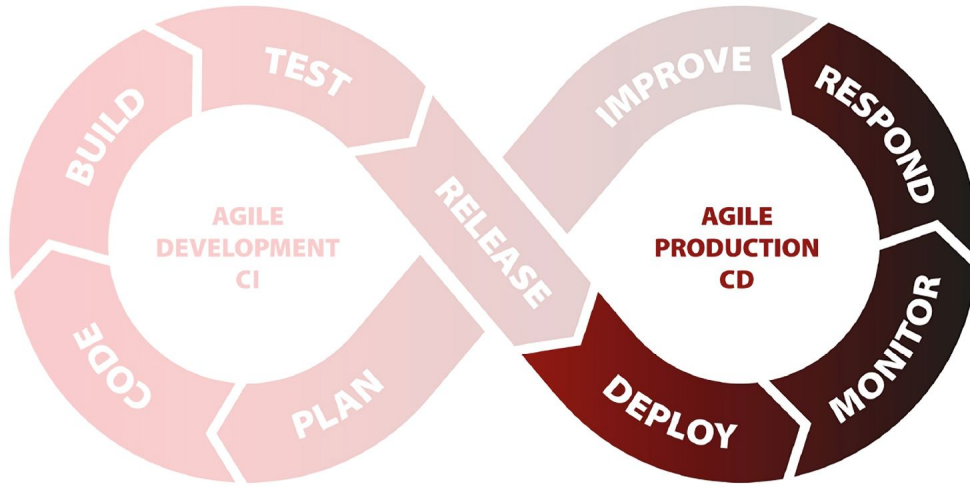


Figure 3.8: DevSecOps Continuum: Deploy, Monitor and Respond phases.

- Use security-approved tools for monitoring the production environment and detect any unusual, suspicious and potentially malicious activity;
- Put in place security-approved processes for the collection and processing of data about customer behaviour, performance, errors, etc.;
- Make sure that the whole team is aware of how to report any suspicious activity to the Product Owner and security teams.

3.2.2 Operating Model

In order to effectively implement the DevSecOps methodology throughout the organization, it is needed to define the involved parties and identify the required people, processes and technologies: all of this must be put together in the definition of an **Operating Model**, which will state how the identified parties will interact one with another and what will be their roles and responsibilities in each phase of the DevSecOps Continuum.

There are several definitions regarding what an operating model is:

- According to Gartner [21], an operating model is *“the blueprint for how value will be created and delivered to target customers”*. In particular, an IT operating model defines how the resources available to the company will be used to achieve the organizational targets;

- The definition reported by the Operational Excellence Society [22] states that *“An operating model is a visualisation (i.e., model or collection of models, maps, tables and charts) that explains how the organisation operates so as to deliver value to its customers or beneficiaries.”*

What all the possible definitions agree on is the fact that an operating model is required by any company that wants to **deliver some value** to their customers and business partners: to do so it is necessary to have an adequate strategy in place, made by some clear objectives that the organization wants to pursue. The aim of the operating model is to combine the high-level company strategy with the available people, processes and technologies, determine the roles and responsibilities of each involved party and how they will interact to achieve the targets defined in the strategy.

As said above, the operating model relies on the three pillars of the PPT framework, which was introduced in Chapter 2 and will now be further analyzed component by component.

People

The people pillar is the one that gets things done, by performing the tasks defined in the processes, possibly leveraging on the available technology.

It is very important for a company to hire the right people, with the correct set of skills that will help with to reach the defined objectives. Along with the technical skills, it is crucial for the success of a project to communicate effectively among team members and to the managers and stakeholders: we cannot get anything done correctly if first it is not understood what it is required to the product or service to be developed.

Roles and responsibilities must be clearly defined inside a team, it will be determining in those moments in which tasks are to be assigned to a given employee and technology is to be selected to support the processes: most of times companies make the mistake of thinking first about the processes to put in place and getting the last generation technology to support them, without thinking about having proper people that can implement and use them. This kind of error brings organization to waste more resources that those that would actually be needed to put in place a process to support a given function.

Once the correct people are found, it is still important to give them proper training: especially in the area of cyber security, training is one of the least expensive

and most effective tools that we have in our hands. Security training is fundamental for every employee of the company and in general for whoever has access to some company asset: the Verizon 2022 Data Breach Investigation Report [23] states that 82% of breaches include human elements, including social engineering techniques, errors and misuses of resources. Training is one of the most effective and least expensive tools that we have in our hands to prevent security issues: employees must be aware of the security threats that could affect the company and what policies and procedures are in place both to prevent and to respond to security incidents.

Processes

Processes define the how a given objective is reached, what actions are needed, in which order, which people has to perform them and what technology is used to support them. Theoretically, the result of a given process should be the same, independently of who performs the single actions.

There are some key aspects to consider while defining a process [15]:

- **People should understand processes**, their steps and, most importantly, their role within it, to do so their tasks should be communicated in a clear way and they should also have a role in the process definition;
- Particular attention should be dedicated first to the **key steps of the process**, those that impact the most on the value to be delivered and the overall efficiency. Once these steps have been properly defined, details and other supporting processes can be defined to improve the performance of the main phases;
- **Metrics** to understand the process performance must be defined and constantly measured and monitored. They are essential to understand how the process behaves and where are the points in which it can be fixed or improved, this is achieved also by constantly collecting feedback from the involved parties.

From the security point of view, it is important to have processes in place to manage the security of the company assets: first of all the organization should have a governance strategy to reduce and manage the risks of unauthorized access or leaks about data and systems, which should include compliance requirements, policies and procedures to assess it. Governance processes should then include analyses of threats, vulnerabilities and risks related to a given company asset and how they can be mitigated or managed and assessments to determine if that asset is the compliant with the company security policies. In the context of secure software development, this translates into security assessments and analyses performed at each stage of the SDLC on the produced artifacts and how they have been produced.

Finally, these processes are a fundamental piece of the DevSecOps strategy, because only by putting them in place we ensure that the application has been developed in a secure way and the result is a robust product that delivers its functionalities while not exposing the user to any security threat.

Technology

The last pillar upon which the DevSecOps Operating Model will be based on is **technology**: companies too often make the mistake to invest lots of money on the latest and most functional tools on the market, considering later the people that will use them and the processes they are intended to support. The investment on technology must be planned after having considered the available skills in terms of people and what are the tasks to be performed by the employed tools inside of the defined processes, otherwise more time and effort will be required to train employees to use the adopted tools and to adapt the process to the tools' functionalities.

When considering technology for a security-related operating model, we must consider all of the tools that will be employed to obtain an overall secure product and secure experience for the final user, by ensuring data privacy and protection through confidentiality, integrity and availability. The main tools that will be considered in this model are tools aimed at assessing the security of the developed product, in terms of AST and other forms of testing and monitoring of the deployed system.

Advantages of Operating Models

Our final target in this work is to define a DevSecOps Operating Model for the development of future Vodafone products and services, based on the strategy which is bringing the Telco company to become a Tech company: the result will be achieved by analyzing each of the nine phases that compose the Vodafone DevSecOps Continuum and identifying for each of them what are the involved people, processes and technologies, what are the roles and responsibilities of each party and how they interact in order to deliver value, which is a secure final product or service for the customer.

In general, there are **several benefits** that come from the adoption of an Operating Model:

- Provides an overview about how the company operates in a given field, what are the involved parties, their roles, responsibilities and how they interact;
- Consistency in the adopted practices, allowing to have excellent results with efficient processes;

- Risks stemming from the individual's perception of a given problem or task are reduced by introducing a standard way to face particular issues;
- Clarification of roles allows a better organization and allocation of the proper resources that are needed for a given project, thus having also a better business performance and easiness of scaling up or down resources if needed;
- Having a precise model for the company internal organization allows to better focus on the customer's needs and the relation with it;
- Employees satisfaction is improved when their role and their impact is made clear and this will reflect also in terms of better performance for reaching the defined targets.

In the following we will analyze the tools, processes and people that will be needed to define a proper DevSecOps Operating Model.

Chapter 4

Tools, resources and processes

In this chapter we will have an insight about all the tools, resources and processes that will be needed to effectively implement the new DevSecOps operating model, stemming from the “people, processes and technologies” idea presented in the previous chapters. We will first have an overview about how we can select the best tools for our purposes and what are the factors and metrics to take into consideration. Then we will dive into the details of the possible tools, with their characteristics and features, of the people that will be involved in the overall process and how they are structured into cross-functional teams and finally the details of the processes that are needed in our new operating model.

4.1 Types of security analyses

The SDLC is a complex process, composed by many phases, each of them producing some kind of artifact, being it a set of documents, diagrams, source code, executables and so on. During and at the end of each phase, security controls can be implemented in order to detect any problem and solve it before the following development phase begins based on the artifact produced by the previous phase: implementing controls at the end of each phase instead of implementing them at the end of the whole process avoids cascading errors and problems created in a given phase to the following phases. This allows to work always with ideally error-free artifacts and to decrease the cost of fixing possible problems since they are solved in the same phase they are created without having to repeat the whole process for an error that was created in the first phases.

4.1.1 Threat Modelling

Threat Modelling is the process that allows identification, classification and analysis of the threats that are intrinsic to a given product, so that risks are evaluated and possibly mitigated. This process is usually carried out in the beginning steps of the SDLC, including requirements and design, where the needed resources are identified and their related risks, for example about sensitive data elaboration. This is a proactive approach to threats and risks mitigation, but reactive approaches like penetration testing (PT) are also needed, as we will see.

There are different approaches to threat modelling, depending on which side we want to focus:

- **Assets:** the target is to identify the most valuable assets and the threats related to them;
- **Attackers:** we want to identify the possible attackers and what targets they could aim to and gain value from, so that also the most valuable assets are identified and protected;

There are several threat modelling methodologies that allow to follow a structured process in order to identify the subject to be protected, the potential threats, the actions that can be taken to mitigate those threats and the validation of such model and taken actions: the most popular methodologies include STRIDE, PASTA, Trike and VAST.

4.1.2 SAST

Static Application Security Testing (SAST) is a white-box testing technique that focuses on the application source code and scans it in order to identify possible security vulnerabilities.

This is a technique used in general early in the development process and helps the developers that are usually focused only on delivering software that meets the requested functional specifications. It can be applied at several levels, which can be function, file/class or application level, what differentiates the levels is the available context: a line of code that could potentially result into a vulnerability in a function level analysis could result as non-vulnerable in an application level analysis, meaning that it would have been a false positive.

SAST has several advantages and disadvantages, the first ones include:

- **Decreased cost of vulnerability fixing:** fixing a vulnerability in development costs approximately 10 times less than in testing and 100 times less than in production;
- **Coverage:** since it is based on source code scanning, it is possible to cover 100% of it to search for vulnerabilities, opposed to DAST that analyses only what is being executed.

The main disadvantages of SAST are:

- **Increased build time:** source code scanning implies a longer build process, which is an idea that is in contrast with the principles of Agile methodologies;
- **False positives:** it is very common that many of vulnerabilities found in source code have no effectiveness when the application is executed because the context of the found vulnerability can never happen, reviewing such false positives in the triage phase is a very time-consuming task and proper tuning of the tools is always needed to avoid as much as possible finding false positives.

4.1.3 DAST

Dynamic Application Security Testing (DAST) is a black-box testing technique that aims at testing an executing application in order to find security weaknesses and vulnerabilities. It can be carried both manually and through automated tools, being the manual analysis more precise but time-consuming and the automated one more efficient but less precise. Being a black-box technique, DAST tools do not have access to the source code, but only at the runtime environment of the application under test.

Vulnerabilities are detected by performing a number of typical attacks on an application aiming at exploiting different kind of vulnerabilities, depending on the type of tested application, such as cross-site scripting, SQL injections, and many more. Tools simulate the attacks and look for unexpected behaviours of the application, which will then be reported to the developers.

Unlike SAST tools, these tools do not depend on the particular language or framework adopted to develop the application, since they analyze its runtime behaviour, and scans can be constantly performed based on the most updated vulnerabilities and attacks.

However, DAST tools should be used carefully in a production-like environment and not in the real production environment itself, otherwise there would be a significant risk of damaging the working application or injecting some malicious

data into the production databases. Also coverage of DAST analyses could be not optimal and the set of provided attacks and possible payloads by the tool may not be the best one for the particular application under test, which would then need particular attention for those test cases not considered by the DAST tool.

4.1.4 IAST

Interactive Application Security Testing (IAST) is a technique that brings together features of SAST and DAST analyses: IAST tools perform analyses in which they are able to correlate the vulnerabilities found in during dynamic testing to particular lines of code, as static testing would do, thus making life very easy to the developers that have to fix the source code in order to remove the vulnerability. To do so, IAST tools clearly have access to both the application source code and its runtime environment.

4.1.5 SCA

Software Composition Analysis (SCA) is an automated process that aims at identifying the open-source components in a codebase, tracking down their vulnerabilities and managing their licenses.

SCA tools analyze every component of a project, source code files, manifest files, package managers, binary files, container images, and so on. The found open source components are tracked down in the Bill of Materials (BOM) and they are compared against the most popular vulnerability databases, such as NVD.

SCA is important because of the very strong usage of open source libraries and dependencies into software products, and tracking them is not an easy task to be performed manually: that is where SCA tools come into help, by finding what known vulnerabilities are present in the developed products and which ones are effectively used into the codebase, thus making it vulnerable and needing for remediation which has to be planned based on the results of the analysis reported by the tool.

4.1.6 VA and PT

Vulnerability Assessment (VA) and Penetration Testing (PT) are activities that must be periodically performed on the company assets and systems in order to find any vulnerability to be possibly exploited by attackers. A company could be equipped with the best prevention and protection systems, such as last generation firewalls, IDSs and IDPs, but even a small configuration error could compromise

the functioning of those systems: this is why periodic VAs are needed to check that the company assets are still not vulnerable to attacks.

VAs can be performed on different types of assets, including:

- Networks and networking devices;
- Servers and hosts;
- Web applications;
- Databases.

When vulnerabilities are found, it is very important to classify them according to their CVSS score, so that remediation actions can be properly prioritized.

On the other hand, Penetration Testing is a technique that aims at legally simulating an attack against an existing system to evaluate if the proper security measures have been implemented and how much they are effective. The tools, techniques and processes used in PT are the same that would be used by a real attacker, usually the attacks that are simulated are the ones that could potentially damage the business and could have legal implications if it is found that the required security measures have not been implemented or are not compliant to the applicable laws and regulations.

PTs can be classified based on various characteristics, such as the type of system to be tested, whether it is performed manually or in an automated way and the amount of information available to the attackers. Based on the latter, we have three types of PT:

- **Black-box:** testers do not know anything about the internal structure of the system, exactly as an external attacker would, so the system is probed for any possible vulnerability;
- **Gray-box:** testers have some knowledge about the internal structure of the system, such as architectural documents, source code, algorithms and data structures, so particular test cases could be designed based on these information;
- **White-box:** testers have access to all of the resources of the project, sometimes including the servers on which the application run. This is the approach that provides the highest amount of security in the smallest time.

Penetration tests are also designed specifically for the particular type of tested system, being it an application, a network, a set of APIs, an IoT device, and many more. They also usually include a series of defined steps, such as:

- **Reconnaissance:** it is needed to recover as much information as possible about the system to be tested, using techniques spanning from non-intrusive network scanning up to social engineering. This is a starting point to determine the possible vulnerabilities and the attack surface;
- **Scanning:** a deeper analysis on the system is performed based on the results of the reconnaissance phase, so that open services, security issues and open-source vulnerabilities are found;
- **Gaining access:** the best technique to gain access to the targeted system is identified, also based on what kind of damage the attacker wants to do to the attacked company, being it data modification, cancellation or stealing or simply damaging the company's reputation;
- **Maintaining access:** it must be checked that once the attacker gained access to the system, it can also maintain it for a time that would allow him to accomplish his goals. This is to demonstrate would could possibly be the impact of a potential attack.

4.1.7 RASP

Runtime Application Self-Protection (RASP) is a technology that allows protection of the system from external threats by leveraging on runtime data and context of the application to be protected, differently from technologies like firewalls that can rely only on network information to detect and block possible attacks. When a possible threat is detected, RASP tools can perform several actions, such as terminating a user's session, sending a warning to the user and alerting the security teams.

The target of RASP tools is to close the gap between the AST techniques and network perimeter controls, which do not have enough control of what really happens in real-time in an application's behaviour and thus do not provide protection against those vulnerabilities that go through the review processes or those threats that were not foreseen during design phases: RASP tools can be compared with IAST tools, with the main difference that the latter's target is to detect vulnerabilities that are inside of the application, while RASP aims at blocking attacks while the application is in its operation phase.

4.2 Tools selection

The selection of the most suitable tools to integrate security controls in an existing development pipeline is a crucial step in the implementation of a DevSecOps strategy: given the facts that there are many different types of security analyses,

each with its own target, and in the set of tools dedicated to a particular type of analysis there are a number of different tools from different vendors, it is important to understand the environment in which we are developing our applications, how they are developed and the level of security that we want and can achieve, so that the proper tools are selected. Also note that the main target of Agile methodologies and CI/CD is to speed up the development and deployment of new functionalities and updates, and security analyses are an additional step that may be significantly time-consuming: thus a balance between the security and efficiency needs must be found.

To approach this phase we need to define some metrics and factors that will influence our choices.

First of all we need to remind that the SDLC is a complex process composed by many phases and each of them takes up some time: security checks, analyses and verifications are additional steps that require additional time, so it is essential to correctly identify the context of the product in order to find a balance between security and performance by applying the correct measures to achieve a good level of security without slowing down the development and deployment processes.

We need to consider many factors when deciding what security measures to adopt: how the application to be analysed is being developed, what tools and languages we are using, what type of application we are creating (web, mobile, etc.), if we are using third-party and/or open-source components, if the development is being completely or partially outsourced, and many more. Identifying and defining these factors will allow us to pick from the huge set only some of the possible security tools, the ones that will effectively improve the quality and security of the application without degrading the performance of the CI/CD pipelines.

For example, if we are partially or completely developing our application in-house, we can have access to source code and apply a SAST tool to perform an analysis to see if there are any known vulnerabilities introduced by badly written code, such as SQL injections, cross site scripting, etc.. Instead for the parts whose source code cannot be accessed, there must be some kind of attestation of the fact that who developed and wrote that source code also tested it with proper tools and it resulted compliant to some defined security standard.

If for any reason we do not have access to the source code of the application to be tested, we can use a DAST tool to perform some dynamic analysis on the running application, trying to exploit possible vulnerabilities and verify if they are there to be eventually exploited by a malicious user who can then perform some kind of attack against our system.

Another factor that can be taken in consideration for almost every system to be developed is the usage of third-party open source components and libraries: being by definition the source code of these elements available to everyone, it is easy for developers to scan those components in order to find if there are any known vulnerabilities. On the other hand, it is also easy for attackers to find possible ways to exploit weaknesses in those components that sometimes are used in many different products developed by very important companies, see the Log4J case that happened in December 2021. It is for this reason that it may also be very important to use a SCA tool to identify and eventually update those libraries that have known vulnerabilities inside them, before they can be exploited causing important damage to the company that used it to implement its products.

Scanlon in [24] proposes a list of factors to keep into consideration while choosing the proper security analyses and tools to perform them, spanning from factors regarding the developed product's nature to the restrictions that are common to the selection of any kind of tool to be used by the company for reaching its business targets.

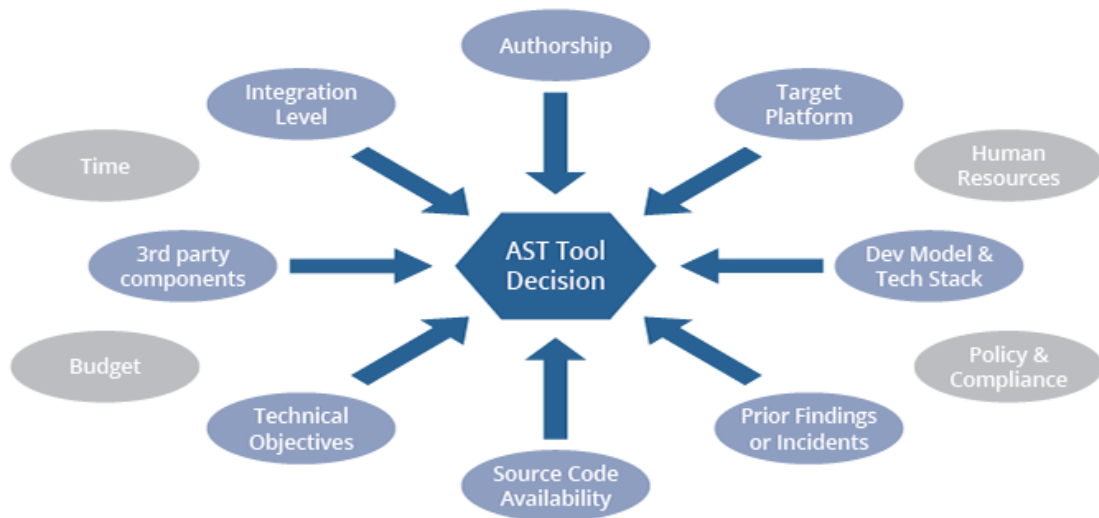


Figure 4.1: AST tools decision factors

As we can see in Figure 4.1 from [24], there are some factors related to the product to be developed and its most suitable types of security analyses and there are other factors related to the particular tool and vendor selection: those factors are the same that are considered for any kind of technology tool selection and are not related to the particular application to analyze.

4.2.1 Factors regarding product type

Authorship

A software product is often a very complex one and there are many possible options regarding its authorship:

- A software can be **completely developed in-house** with all of its modules and dependencies, so that all of the source code is available to be scanned for vulnerabilities. This is an approach that can usually be afforded only by very large software companies;
- On the contrary, a software product development can be **completely outsourced to an external third party**. In this case the product can be then either delivered as an executable only or along with its source code, with all the limitations of the case. This may be the case for a small company that needs a product with some given characteristics, but that has not the resources to develop it in-house;
- Most of the times a software product is given by the **combination of the previous** two options: a company has the resources to develop some modules of the product, but will have to rely on external modules to implement some functionalities. Those modules could be either open source or proprietary, meaning that in some cases a deeper analysis using SAST tools can be performed and in other cases we will have to rely on other methods.

It is clear that the best scenario would be the one in which all of the suitable and possible security analyses are performed, but in case that for some reason (budget, efficiency) it is possible to choose to use only one type of security tool we must make sure to use the best tool given our context:

- If all or most of the application was developed in-house, we should choose a SAST tool, which will perform the analysis that gives out the most accurate and detailed results about security issues;
- Instead, if the application comes from a third party, a DAST tool could be the choice to test the provided executable for vulnerabilities that could be exploited at runtime. The usage of SCA tools should also be considered, if many open-source external libraries have been used. Also, we may consider the possibility of asking the third party that owns the source code to provide some kind of certification of the fact that the code has been security tested and does not contain any kind of flaw or vulnerability.

The below flowchart in Figure 4.2 represents the steps to consider when choosing the proper security tools.

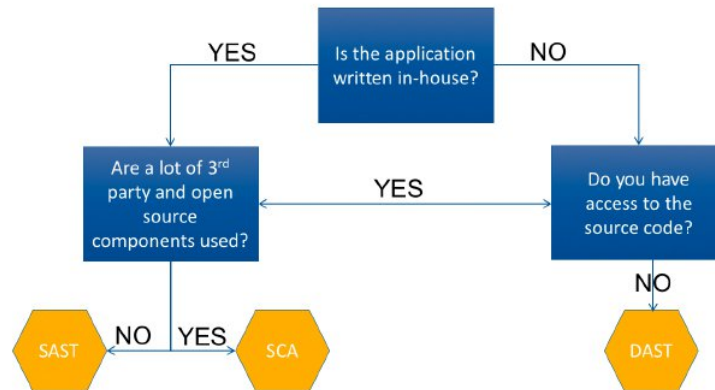


Figure 4.2: AST decision flowchart

Source code availability

This point is strictly related to the previous one, since source code availability mostly depends on the authorship of the application, apart from its eventual open-source components.

If source code is not available, we can rely mostly on DAST tools and techniques such as fuzzing and negative testing:

- **Fuzzing** is a technique that aims at testing possibly all of the possible inputs that an application can receive and the application's behavior in response to these inputs. Most of the times it is not feasible to test every single input of the application because it would take too much time, so only significant inputs are taken into consideration;
- **Negative testing** instead aims at testing the handling of exceptions, some unexpected input is provided to the application and it must be handled in a graceful way, without causing the application to crash or leak some sensitive data.

If source code is available, the best choice is to perform SAST analysis. If the external factors allow to, it would be even better to apply both SAST and DAST tools: in this case we may consider using IAST tools, which analyze the runtime behavior of the application but can also correlate it with the proper part of the source code, so that we have a better identification of the vulnerability and how to fix it. Build-and-compare tools could also be a good option: with these tools

we can check if the delivered source code can be built to get an executable that is identical to the provided executable so to ensure that the external party did not inject anything after compiling and building the source code.

Third-party components

As previously mentioned, it is very common that an application makes use of external libraries to add some new functionalities and features: those third-party components could contain vulnerabilities that make the products which make use of them exposed to different kinds of attacks, and if the vulnerable library is a very popular and extensively used one then there could be very important problems.

As a first choice, if the developed application makes an extensive use of third-party components and libraries, having a SCA tool to run an analysis on the product could be more important than a SAST tool: SCA tools analyze what libraries are used in the project and informs us about any vulnerability that has been found in them by consulting the most updated and important databases containing information about known vulnerabilities in software products, such as the NVD.

If we consider also the factors mentioned up to this point, the suggestion would be to use a combination of SAST and SCA tools if we have extensive access to the used source code and several external libraries were used, or use DAST and SCA tools if the application was developed by a third party and we want to make sure that no vulnerable component has been included in our product.

Development Model

The methodology that was chosen to manage the lifecycle of the product also impacts on the choice of the most suitable type of security analysis:

- In a standard waterfall SDLC model, with long and sequential phases, SAST and DAST tools fit well and they should be integrated as soon as possible in the development process;
- In an Agile model, possibly using CI/CD pipelines and/or a DevOps methodology, IAST and hybrid tools may fit better since they combine the characteristics of SAST and DAST analyses, which could be too time consuming in a process where the efficiency and the speed in delivering new features and updates are essential.

Target Platform

The type of platform on which the product will be used also allows to select the proper tools and techniques to perform AST:

- If the application is to be used on mobile devices, then MAST (Mobile Application Security Testing) techniques should be used, focusing attention on those security issues that are particular to the mobile environments;
- If the application is to be exposed on the Internet, then DAST tools could be very useful to identify vulnerabilities that could be exploited by black-hat hackers worldwide;
- If the application is to be deployed on a cloud environment, then we may consider ASTaaS (AST as a Service), which could be easily integrated in most cloud services.

Integration level

This factor relates to how early in the development process can AST tools be integrated: what can be applied here as a general rule is the shift-left idea, that is, when possible, we should integrate security analysis tools as early as possible in the software lifecycle, so that bugs, flaws and vulnerabilities can be located and resolved before proceeding to the subsequent phases and environments and they become increasingly expensive to fix.

Compliance

Depending on the target of the developed product and its context of application (e.g. healthcare, banking, etc.), it may be necessary to comply to some policies and standards regarding information security and privacy: these include regulations defined by the government bodies of the countries that companies work in, or may be internal policies and guidelines defined by the company and to which the employees and the products they work on must comply to.

For policies and regulations that are internationally known and adopted, AST tools may have already some integrated features that allow developer companies to be compliant to such regulations. Instead for complying to internal policies, some more custom rules and controls may be needed and those need to be defined and integrated in the overall process.

If there are no particular policies to follow, developers may think to follow some standard lists such as the OWASP Top 10, SANS Top 25 or CERT Coding Standards in order to avoid vulnerabilities as much as possible.

Since compliance mostly regards how data are elaborated and stored and this is why database security scanners may be useful in this case. Also, there may be cases in which the rules to be followed become very complex and numerous, so we may need correlation and ASTO (AST Orchestration) tools to have a complete overview of what is compliant and what needs to be fixed and how.

Prior Findings and Incidents

It is very important for a company to keep track of the past issues and incidents, so that lessons can be learnt, and countermeasures can be adopted in order to avoid having again the same issues:

- If we know that a known vulnerability of a third-party component included in our product has been exploited, consider using or improving the usage of a SCA tool;
- If code reviews point out bad coding practices, consider adopting more strict rules in SAST tools so that vulnerabilities embedded in source code are spotted and resolved before that goes into production.

Summary

As already said, the factors which have been just presented have to be considered altogether: of course an ideal solution would include all of the different types of analysis in order to eliminate as many vulnerabilities, bugs and flaws as possible, always remembering that an absolute absence of security issues can never be achieved, thus the adoption of the measures mentioned so far does not exclude the adoption of further tools to ensure a higher level of security.

The considerations that we just made have been made to help prioritizing one type of analysis over another in the case in which for any reason it is not possible to integrate all of them in the development process, so that the security of the product is always optimized to the best of possibilities. As a general rule it is always better to first understand and integrate those tools that perform analysis common to all types of application, independent of their target platform, that is SAST, DAST and SCA tools, adding eventually IAST and hybrid tools that help conduct these types of analyses. Then, on top of these tools, further tools

targeted to the particular type of application can be considered, such as MAST, ASTaaS, etc.. Finally ASTO and correlation tools can be very useful to integrate the results of the different analyses coming from different tools in order to have a clear picture about the present vulnerabilities and how they can be fixed and so planning and prioritizing proper remediation actions. In Figure 4.3 we can see this concept represented as a pyramid where the most general security tools are its base and on top of that we put other tools to create a more complete and secure development environment.

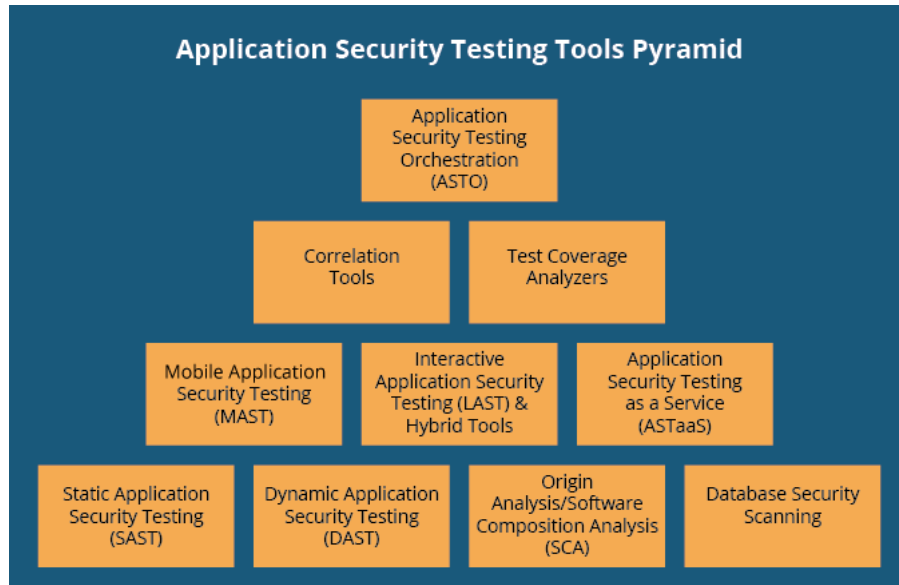


Figure 4.3: AST tools pyramid

4.2.2 Factors regarding tool selection

Budget

Depending on the available budget, we may have to choose a limited number of commercial products to be bought, thus it is important to identify and prioritize the analyses to be performed, being it SAST, DAST or SCA. A compromise could be found in the usage of a IAST tool, that brings together features from both SAST and DAST and would avoid the purchase of more tools that may also need to be integrated. ASTaaS could be also an option that may be expensive at first, but in the long run it could result to be a cheaper option that allows to pay only for what is effectively used instead of paying a monthly or yearly fee and can be easily integrated in cloud environments.

We must not forget that there are also many free and open source products which should be tried out before choosing to buy a commercial solution: trying these solutions could help in understanding what are the aspects to be prioritized and if some of the budget could be saved by finding a solution that satisfies the business requirements without having to buy an additional product.

Development Language and Technology Stack

We have to consider the already existent and used programming languages and technology stack, which cannot be easily replaced: it is important to find security analysis tools that are compatible to the used development language, so that source code and libraries can be properly scanned for vulnerabilities. There are many tools for popular languages such as Java, .NET, C/C++ and many more, so it is only a matter of choosing a tool that supports the languages used in the developed product.

It is also important that the selected tools can be easily integrated with the development tools that are currently used in the different phases of the SDLC: for example it could be very useful to have a SAST tool that scans code as soon as it is pushed into the code repository and so it is important to have integration between the chosen security tool and the already used code repository tool. A similar idea can be applied to IDEs and their plugins: as the shift-left approach suggests, it would be great to have a tool that points out vulnerabilities as soon as they are introduced in the source code, this would help the developer both to not add flaws into the application and to develop a more advanced security culture that in the long run will help to develop more overall secure applications.

Technical Objectives

In the product requirements there may be some technical requirements such as controls related to particular kinds of vulnerability like SQL injections, cross-site scripting, input validation and sanitization, ACLs and access management, etc.: we must make sure that the adopted AST tools meet the requirements of the developed products.

Time and Human Resources

The final target of the implementation of security tools in the development pipeline is to save time, effort and money by not having to rework on past phases' results to fix problems that were found later in the process: but it must also be considered

that the initial implementation and integration of AST tools requires time and effort by specialized people that need to learn how the tool works, how to properly integrate it and tune it to the specific requirements of the product, in order to avoid false positives and false negatives and always improve the finding and remediation of vulnerabilities.

4.2.3 Tools technical factors

There are also some technical factors related to the particular tool selection that we may want to consider such as:

- the coverage of the implemented controls in terms of lines of code analyzed for a SAST tool;
- the amount of possible inputs to the application to be tested evaluated by a DAST tool;
- the efficiency of the analysis in terms of time and speed, this is a particularly relevant factor in the context of DevSecOps since one of the main targets of this methodology is to deliver software in a fast way;
- and many other factors that may change depending on the requirements of the business and the required level of security to be achieved by the developed product.

A very important factor that can be very appreciated by the professionals that have to use every day these AST tools is the user experience: it is often underestimated, but for an individual that has to analyze possibly many applications every day it is very important that the least possible amount of time is spent on understanding which problems are worth of attention and which are not, and for this purpose a tool should provide good reporting capabilities and dashboards that allow to quickly prioritize and schedule remediations for any type of problem that may arise from the security analyses.

4.3 Tools

We will now present some of the tools that have been selected to be used in the DevOps pipeline, starting from the already present on-premises tools, up to the those that will be used in the cloud-based pipeline. Each phase of the DevOps process has a set of tools that can support it and that we can pick one or more from, on the other hand a single tool may support one or more phases of the development process.

4.3.1 Pipeline on-premises

Jira

Jira is a suite of proprietary products produced by Atlassian, what brings together all of them is the target of making the work planning easier, being it a software development work, an IT service desk work, and many more.

Git, GitHub and GitLab

Git is a free and open source distributed Version Control System (VCS) designed to have fast and easy code management in large projects, having programmers that collaborate on many different branches of a single project: as other distributed VCSs and unlike client-server systems, each Git directory on local computers is a repository with complete history and version-tracking abilities, independently of network access.

There are several hosting services that allow to create and manage Git repositories online, among which we find GitHub and GitLab.

GitHub is a platform, currently owned by Microsoft Corporation, that is mainly thought to allow open-source projects sharing and collaboration among many users through issue tracking and pull requests that help the owner to fix bugs thanks to a global community. It also helps developers to maintain a project history and track its changes.

GitLab is a platform developed and owned by GitLab Inc., open-source for its community edition, that offers various features for a complete management of the software development lifecycle, since it can be easily integrated with several tools performing different functions and it is a highly secure cloud-native application.

Maven

Maven is a build automation tool produced by the Apache Foundation that offers several functions to automate the build process making life easier to the developer and the process itself less error prone.

Some of the phases that are handled by Maven are code compilation, binaries packaging, running the tests on the source code, deployment on systems and finally the project documentation. One of the tool's main components is the Project

Object Model (POM) that is stored in the pom.xml file and defines the structure of the project, its dependencies, tests, documentation, and so on.

Jenkins

Jenkins is an open-source tool written in Java that provides CI/CD features, it is both distributed as an application to be executed on a web server and then accessed remotely through a web browser and a Docker image to run in virtual environments.

It enables automation of various phases of the SDLC, from compilation up to deployment, and tasks, such as code commit to the repository or tests execution, can also be scheduled to be triggered at a given time. This is done through the definition of pipelines, which are composed by one or more stages: each stage ideally should correspond to a step of the development process and we can define what is the behaviour at the end of each stage depending on its success or failure. Pipelines can be defined both through code and through a graphical interface.

A big plus of this tool is the wide selection of plugins developed by the community that allow easy integration of many different tools in the development pipeline.

Nexus

Nexus Repository is a tool that allows build artifact management through a central repository from which we can retrieve any kind of stored artifact, such as binaries, components, libraries and containers, whenever they are needed, for example in the build process of a product. The artifacts in the repositories are versioned as in other types of repository managers and they provide a single point of truth for anyone in the company.

This tool can be integrated with the most popular IDEs and CI/CD tools, such as Eclipse, IntelliJ, Jenkins and Visual Studio.

Docker

Docker is an open-source tool that allows creating, testing and distributing applications in a very easy way by using containers: compared to virtual machines which need an operating system each, Docker allows to deploy several containers upon the same OS, what is only needed to be run is the Docker Engine, which will be

able to instantiate and execute the containers.

What Docker creates are Docker images: each image has everything that an application needs to run, including its code and dependencies, and is a read-only file. Containers are live running instances of Docker images that operate on temporary data that exist only as long as the container exists.

Containers are much easier and faster to deploy than virtual machines, for this reason they are a solution that is increasingly being used in Agile development models.

Kubernetes

Kubernetes is an open-source container orchestration and management platform, working with many containerization tools, including Docker. Initially developed by Google, it is now maintained by the Cloud Native Computing Foundation.

The main purpose of this tool is to manage distributed environments and their resources making containerized applications always available, this is done through a number of features, such as:

- load balancing on different containers if the traffic on one of them is too high;
- storage orchestration that allows choosing what kind of storage to mount (local, public cloud, etc.);
- load optimization performed by providing to Kubernetes a cluster of nodes with a given amount of resources, the system will then optimize the available resourcing by instantiating the proper number of containers;
- self-healing when a container freezes it is replaced and the broken one is terminated, same with containers not responding to health checks, in order not to waste resources.

The desired state for the containerized application is described into configuration files, in YAML or JSON format, that are read by the machine and translated into proper configurations, making the system go from the actual state to the desired state with a series of transitions.

Openshift

Openshift is a containerization Platform-as-a-Service (PaaS) which is based on Kubernetes plus some components that add functionalities, such as Source 2 Image, the technology that takes an application's source code and returns the containerized application, and Jenkins.

It is a product thought to be easy to use, it provides both a web-based interface and a CLI and a set of APIs that allow management of every aspect of the platform. It provides a very rich set of languages and servers to build complex distributed systems and microservices-based applications in an easy way.

This tool allows both developers to build autonomously sophisticated workflows for building and deploying applications and operations people to have full control on the used resources and the overall environment.

Junit

Junit is a framework for unit testing of Java applications, it had a very important role in the definition of the Test Driven Development (TDD), emphasising the importance of tests in the development of stable software and reduction of time-consuming debugging activities later on.

Ansible

Ansible is a software that enables automation of the configuration and management of Unix-like and Windows systems. As in most of these configuration software, Ansible is based on two types of machines, which are nodes and control nodes: the first ones are controlled by the latter only through an SSH agent, since Ansible is an agentless solution, unlike many of its competitors, making it very easy to deploy it and to configure the nodes without overloading the network. Nodes are managed by installing and running modules: when these processes are running on the target machines, a JSON-based protocol is used to exchange messages between the node and the controlling node, while when this process is not running Ansible does not use any resource of the node since it has not an agent running on it.

Selenium

Selenium is a range of tools for functional testing in web browsers, it allows authoring tests through Selenium IDE and simulating the behaviour of a real user through Selenium WebDriver and Selenium Grid. It provides also a domain-specific language (DSL) called Selenese to write tests for a number of popular programming languages, including Java, Python, PHP, C and JavaScript.

JMeter

JMeter is an open-source project of the Apache Foundation that allows performing load tests mainly on web applications and the servers hosting them, in order to determine if that application is ready to endure high traffic loads for example. It is used thanks to its possibility to integrate these tests in the already existing ones through specific APIs, but it also provides a graphical interface to set up the tests.

4.3.2 Additional tools for cloud migration

AWS

Amazon Web Services (AWS) is a company of the Amazon group that offers cloud computing services, with over 200 offered products. Their services are offered on an on-demand basis, with high availability, redundancy and security features: the final cost of the used services is based on the combination of the type and quantity of utilized resources, usage time and desired performance.

Terraform

Terraform is an open-source Infrastructure-as-Code (IaC) software tool created by HashiCorp that allows defining a desired data center infrastructure through a declarative configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON. The scripts describe the desired final state of the infrastructure and that is reached through a series of CRUD actions performed on user's behalf. It supports a number of cloud infrastructure providers, including AWS, and one of its main benefits is the fact that configuration scripts are independent of the cloud platform we want to integrate with, making them very portable.

Datadog

Datadog is a cloud infrastructure monitoring service, providing dashboards, alerting systems and visualization of metrics. It supports the main cloud providers, including AWS, Google Cloud Platform, Microsoft Azure and RedHat OpenShift.

4.3.3 Application security tools

SonarQube

SonarQube is an **automated code review tool**, born first mainly for code quality analyses (finding bugs, code smells, etc.), and later implementing security analyses features, thus allowing to perform **SAST** on source code files for a number of different programming languages in order to detect and fix possible vulnerabilities. It can be easily integrated with the most popular build tools, such as Maven and Gradle, and continuous integration tools, such as Jenkins.

Code analysis with SonarQube is one of the steps of the common development process, as we can see in Figure 4.4.

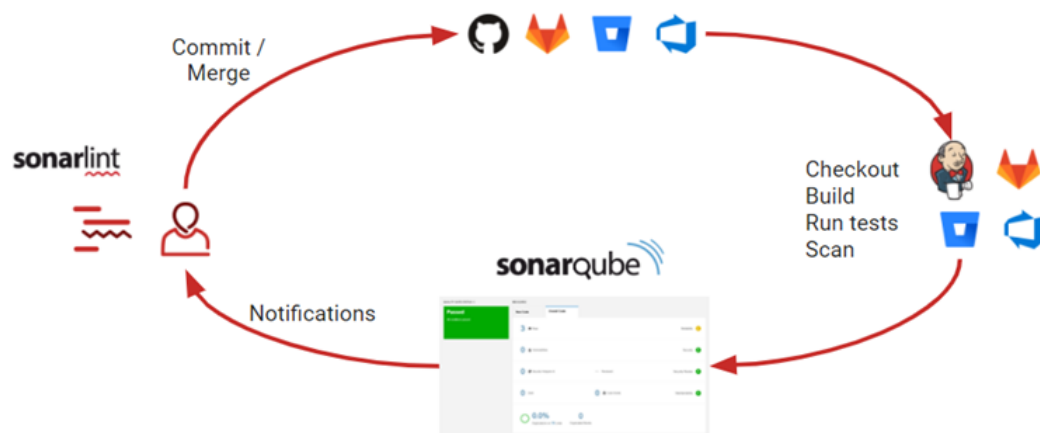


Figure 4.4: SDLC schema including SonarQube

The cycle starts with the developer writing code into the chosen IDE (possibly equipped with the SonarLint plugin, see later). Code is then committed and pushed onto the repository, and finally merged when the work is completed. The files are then checked out by the continuous integration tool, the build of the project is performed, unit tests are run and finally the SonarQube scan is run. The scanner then publishes the results on the SonarQube server, so that they can be analysed from the web interface and/or developers can receive notification through e-mail or

directly on their IDEs: based on these results, either some remediation action may need to take place or the developer can continue his/her work on the source code.

By giving an almost immediate feedback to the developer, SonarQube allows to adopt a Clean As You Code approach that makes code quality to be the best as possible: by focusing on new code anytime a scan is performed, the developer can easily fix the issues as soon as they arise and always deliver high quality code. The SonarQube features that make adopting this strategy possible are mainly the Quality Profiles and Quality Gates.

We will now give an overview of the tool and its features by looking at the web interface, starting from the Projects homepage.

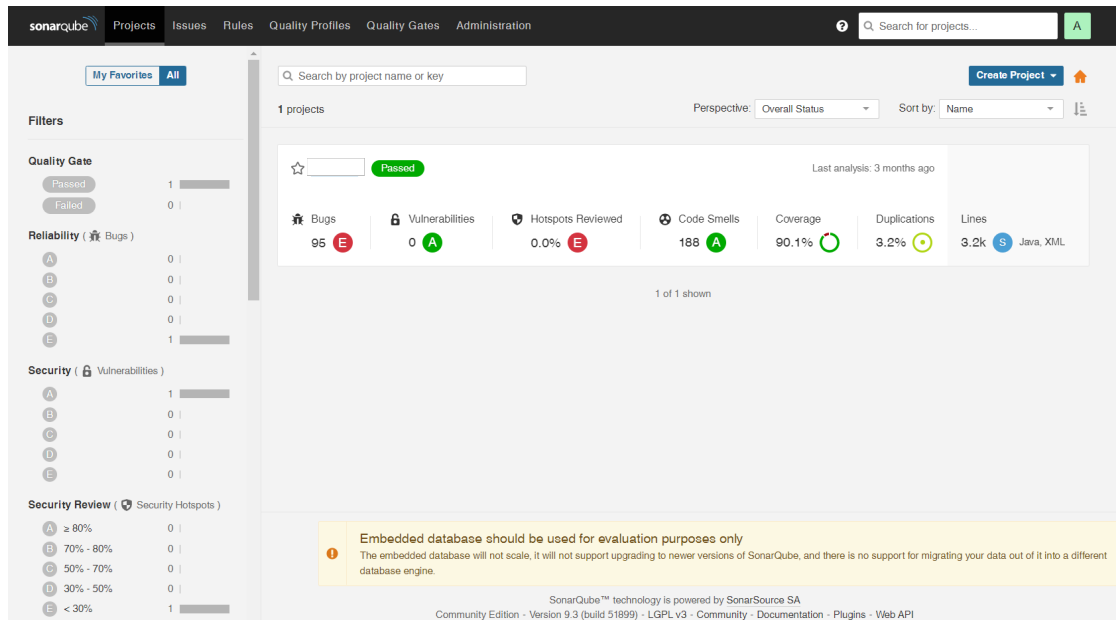


Figure 4.5: SonarQube web interface homepage

As we can see in Figure 4.5, in this page we can see all the projects that have been onboarded onto the platform and analyzed at least once and apply filters based on some metrics, such as projects having a given rating for Reliability, Security, Security Review and Maintainability, size of the project, coverage of the tests, and so on. Project can of course be created mainly in two ways:

- **manually**, by choosing the desired CI platform in which we want to integrate SonarQube, being it Jenkins, GitHub Actions, Bitbucket Pipelines, GitLab CI, Azure Pipelines or other CI platforms. We have also the possibility to

onboard a local project for testing or other particular cases;

- **from a DevOps platform**, such as Azure DevOps, Bitbucket, GitLab or GitHub, this allows to benefit from the repository import and pull request decoration features of SonarQube.

If we go into the project details, we find a view like the one in Figure 4.6.

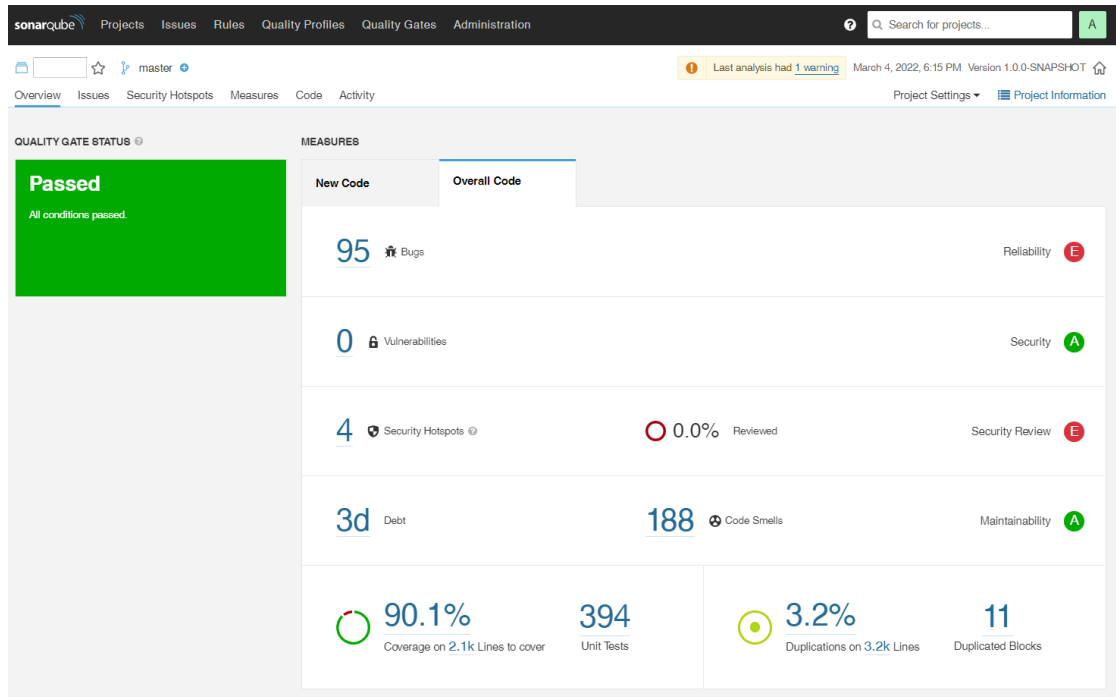


Figure 4.6: SonarQube project overview

Here we find an overview of the results of the source code analysis performed on the project: we have two views, one on the New Code, which reports the results of the analysis on the code that has been added or modified since the last analysis, and another on the Overall Code, which reports the results on all of the source code of the project.

There are a few metrics and measures that are evaluated by SonarQube on the project and for some of them a rating provided as a grade from A to E is given. The evaluated metrics are:

- **Bugs:** evaluates the total number of bugs found in the source code, based on this metric a rating about the Reliability of the project is given;

- **Vulnerabilities:** evaluates the total number of vulnerabilities found in the source code, based on this metric a rating on the Security of the project is given;
- **Security Hotspots:** these are particular parts of the code which are deemed to be security sensitive and need to be reviewed by the developers and the security team in order to determine if this is an issue to be solved or a false positive. Independently of the result of the review (which can be Fixed or Safe), in the end the hotspots must be marked as reviewed, so that SonarQube keeps track of it and gives a rating on the Security Review of the project;
- **Code Smells:** these are not bugs or errors in the code, they are issues that make the code less maintainable, they may be due to a bad design and may make life difficult to the developers that in the future will have to work on this part of the code, because at best they will have a hard time in understanding the “smelling” code and waste time, but in the worst case they may become a source of bugs and vulnerabilities. It is for this reason that Code Smells are an indicator for the Technical Debt, that is the time that will be eventually spent in fixing these issues, and based on these measures a rating on the Maintainability of the code is given;
- **Coverage:** this is a measure of how much of the lines of code of the project are tested by the provided unit tests, whose number can be also seen in this view. The more of the source code is covered by the tests, the more accurate the eventual remediation actions can be;
- **Duplication:** blocks of duplicated code may also be an issue for the maintainability of the source code, so it is always best to keep this percentage and the number of blocks as low as possible.

The ratings seen above are given based on precise thresholds for each of the characteristics, as we can see in the below tables.

Rating	Threshold
A	0 Bugs
B	At least 1 Minor Bug
C	At least 1 Major Bug
D	At least 1 Critical Bug
E	At least 1 Blocker Bug

Table 4.1: Reliability ratings

Rating	Threshold
A	0 Vulnerabilities
B	At least 1 Minor Vulnerability
C	At least 1 Major Vulnerability
D	At least 1 Critical Vulnerability
E	At least 1 Blocker Vulnerability

Table 4.2: Security ratings

Rating	Threshold
A	At least 80% reviewed
B	From 70% to 80% reviewed
C	From 50% to 70% reviewed
D	From 30% to 50% reviewed
E	Less than 30% reviewed

Table 4.3: Security Review ratings

Rating	Threshold
A	Ratio from 0 to 0.05
B	Ratio from 0.06 to 0.1
C	Ratio from 0.11 to 0.2
D	Ratio from 0.21 to 0.5
E	Ratio from 0.51 to 1

Table 4.4: Maintainability ratings

If we switch to the Issues tab of the project we will find a page like the one in Figure 4.7.

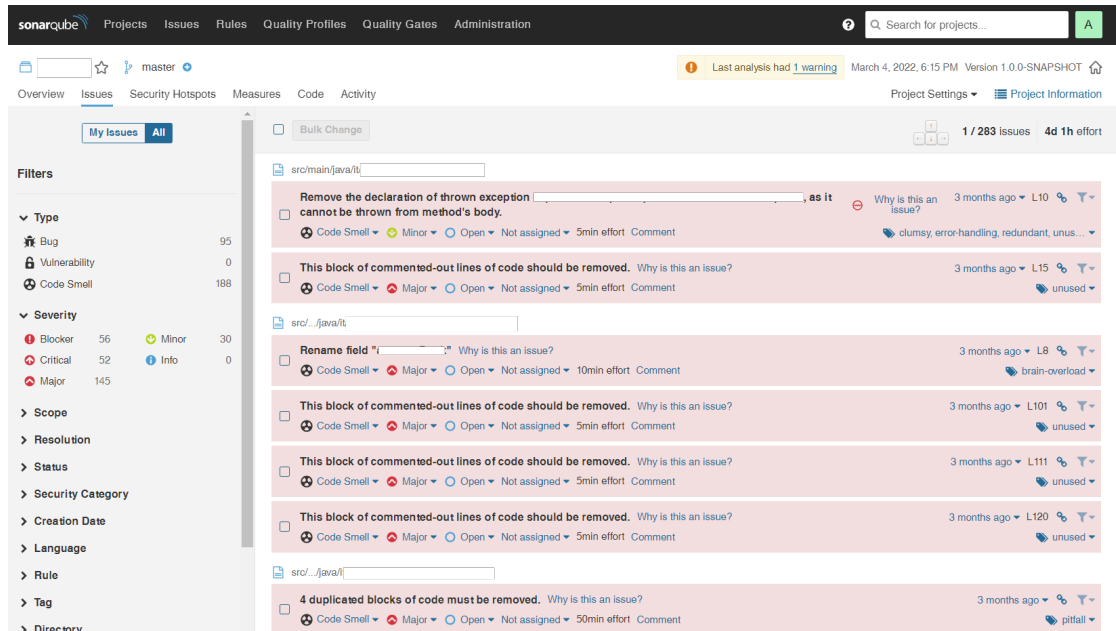


Figure 4.7: SonarQube project issues

Here we can see the details about all of the Bugs, Vulnerabilities and Code Smells found in the project: on the left we see all of the filters that we can apply on the issues.

The main features of the issues are:

- **Description:** explains the issue and a small suggestion about how to solve it;
- **Severity:** there are five levels of issue severity, as reported in the SonarQube documentation [25]:
 - **Blocker:** issue that may impact the application in production with a high probability, must be immediately fixed;
 - **Critical:** issue with a low probability to impact the application in production or a security flaw, must be immediately reviewed;
 - **Major:** quality flaw that may highly impact the developers productivity;
 - **Minor:** quality flaw that may slightly impact the developers productivity;
 - **Info:** neither a bug nor a quality flaw, just a finding.
- **Status:** each issue has a lifecycle composed by five statuses:

- **Open:** status set automatically by SonarQube upon issue creation;
 - **Confirmed:** status set manually to indicate that the issue is valid;
 - **Resolved:** status set manually to indicate that the issue status should be set on Closed upon following analysis;
 - **Reopened:** status set automatically by SonarQube when a Resolved issue is not actually resolved;
 - **Closed:** status set automatically by SonarQube when an issue has been resolved.
- **Resolution:** an issue may have been resolved in different ways and depending on its status may have different Resolutions. Closed issues may be Fixed or Removed, while Resolved issues may be False Positive or Won't Fix;
 - **Assignee:** each issue can be assigned to a particular user to be resolved;
 - **Time effort:** an estimate of the needed time so resolve the issue.

The two main components on which SonarQube analyses are based on are Quality Profiles and Quality Gates.

Quality Profiles are sets of Rules, defined for each language, that are used during analyses of a project: rule violations will raise Issues. SonarQube defines a default Quality Profile for each supported language and ideally that profile should be used for analyses of all project developed using that language, but custom Quality Profiles can be defined based on the particular project's features to be analysed.

The page for defining Quality Profiles is the one we see in Figure 4.8.

Quality Gates are sets of conditions that will determine whether a project is ready for release or not: each Quality Gate is composed by one or more threshold on a given metric, if at least one of these thresholds is crossed, the Quality Gate fails and the project build is stopped.

Thresholds are made by three components:

- A metric;
- An operator;
- A threshold value, that can be either an absolute quantity or a percentage.

Quality Profiles

Quality Profiles are collections of rules to apply during an analysis.
For each language there is a default profile. All projects not explicitly assigned to some other profile will be analyzed with the default. Ideally, all projects will use the same profile for a language. [Learn More](#)

Filter profiles by:

C#, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way BUILT-IN	DEFAULT	255	3 months ago	Never

CSS, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way BUILT-IN	DEFAULT	23	3 months ago	Never

CloudFormation, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way BUILT-IN	DEFAULT	26	3 months ago	Never

Flex, 1 profile(s)	Projects	Rules	Updated	Used
Sonar way BUILT-IN	DEFAULT	47	3 months ago	Never

Recently Added Rules

Source files should not have any duplicated blocks
C#, not yet activated

Skipped unit tests should be either removed or fixed
C#, not yet activated

Lines should have sufficient coverage by tests
C#, not yet activated

Source files should have a sufficient density of co...
C#, not yet activated

Failed unit tests should be fixed
C#, not yet activated

Branches should have sufficient coverage by tests
C#, not yet activated

Lines should have sufficient coverage by tests
HTML, not yet activated

Source files should have a sufficient density of co...
HTML, not yet activated

Source files should not have any duplicated blocks
HTML, not yet activated

Skipped unit tests should be either removed or fixed
HTML, not yet activated

[See all 2.7k](#)

Figure 4.8: SonarQube Quality Profiles

Quality Gates

Sonar way **BUILT-IN** [Copy](#)

Conditions

Conditions on New Code

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Projects

Every project not specifically associated to a quality gate will be associated to this one by default.

Figure 4.9: SonarQube Quality Gates

The example in Figure 4.9 is the SonarQube default Quality Gate, but custom Quality Gates for our projects can be defined.

Conditions to the Quality Gate can be added by choosing the metric we want to consider and the desired threshold, like in Figures 4.10 and 4.11 below. The condition can apply either to the New Code only or on the Overall Code.

Add Condition

☒ On New Code ☐ On Overall Code

Quality Gate fails when

Search for metrics...

Figure 4.10: SonarQube Quality Gate condition, metric selection

Add Condition

☒ On New Code ☐ On Overall Code

Quality Gate fails when

Vulnerabilities

Operator

is greater than

Value

0

Figure 4.11: SonarQube Quality Gate condition, threshold selection

Mend SCA

Mend, formerly known as WhiteSource, is an application security company that provides different kinds of solutions for securing applications, starting from their development: indeed, it provides SAST, SCA and other solutions to be integrated in the SDLC and the different adopted tools.

Here we are going to focus on the **Mend SCA solution**, which allows to perform security analysis on the third-party and open-source libraries and components that are integrated in a product: this tool will analyse our product and report if there are any known vulnerabilities in the used libraries, to do so the Mend Vulnerability Database is built and consulted: this database is built starting from the most important vulnerability databases, with NVD above all, but Mend takes into consideration the facts that:

- the open-source world is very decentralized and many times it could be very difficult to find new vulnerabilities;
- about 20% of the publicly disclosed vulnerabilities are not included in the NVD.

So, the research team at Mend came up with methods to identify vulnerabilities in open-source components as soon as they are created, to include them in their vulnerability database and finally allow the SCA solution users to know as soon as possible if their projects' external dependencies include any vulnerability that could possibly need remediation in a very short time. This is made by indexing many sources apart from NVD and analysing billions of open-source files, millions of open-source libraries and many package managers, technologies, languages, and so on.

To make life even easier for developers and security teams, Mend provides also a feature called **Effective Usage Analysis (EUA)** that reports if vulnerabilities present in the external libraries are also actually present in the source code of the analysed project, thus excluding false positives and giving a better overview on what are the needed remediation actions and how they need to be prioritized, in order to avoid the product from being vulnerable to any kind of attack.

One of the main features on which the software composition analysis is based on is the Bill of Materials (BOM), which in the case of software products becomes the **Software Bill of Materials (SBOM)**. The BOM is a concept that is widely adopted in all manufacturing fields and it is a list of all the components, materials, parts, etc. which are needed to produce the final product, it is a very important element used in supply chain management (SCM): in the field of software engineering, we do not have physical parts or components to be assembled to build a machine, instead we have many software libraries and tools, that can be either open-source or proprietary, which are being used together in order to produce the final product. The SBOM is used to track every component used in the development of the software product, and that turns to be useful both for the producers and the customers:

- the producer can use the SBOM to easily identify the components which are out of date or vulnerable and need to be updated;
- the consumer can use the SBOM to perform vulnerability and license analysis, that can be useful for risk management.

SBOMs are used to prevent supply chain attacks, which are a major threat that software producers have to deal with: a study from Argon [26] states that the amount of supply chain attacks tripled in 2021 with respect to 2020, this is due to mainly three factors:

- **vulnerabilities in open-source components**, which could be either accidentally inserted and then found and abused by attackers (see Log4j) or

inserted on purpose by poisoning libraries prior to being downloaded and used (see `us-parser-js` poisoning);

- **compromised pipeline tools**, with the increasing usage of automated pipelines for software delivery, also the number of tools being used is increasing, thus the probability to find a vulnerability to be exploited in these tools, being it a defect of the product itself or a misconfiguration;
- **upload of bad source code into repositories**, this reflects into bad quality artifacts which are full of defects and require lots of time to be fixed, thus decreasing productivity.

Mend provides a solution [27] for companies to easily produce a SBOM to include in their products, which is also a US federal government requirement for every software product: what is required is a formal, machine-readable inventory of all components and dependencies used in building software. The Mend SBOM not only lists all the components that a software is made of, but also provides remediation paths for those components that need to be updated due to present vulnerabilities. It is produced in the form of a SPDX document, which is machine-readable, and provides an inventory of all the software components, their dependencies and their hierarchical relationships.

The Mend SCA solution provides different types of interfaces and ways to be integrated in the current development pipeline: it provides a web interface with very good dashboards and reporting tools that provide any kind of data on the analyses performed on the projects and it can be easily integrated with the most used IDEs, such as Visual Studio, Eclipse and IntelliJ IDEA, and code repository platforms, such as GitHub, GitLab and Bitbucket.

We now analyse the main features of the web interface, starting from the product main page: each product onboarded on the Mend SCA platform has a dashboard as the one we can see in Figure 4.12 providing a high level overview about the results of the last analysis performed on it.

In this page we can find a very rich dashboard, reporting several kinds of information about the product under different forms. In the bottom-left part we see two tabs about the composition of the product:

- **Project Summary**: this tab reports all of the projects that compose the product and the number of libraries included in each product;
- **Libraries**: here are reported the number and the list of all of the libraries included in the product, with their name, the license(s), the number of projects in which they are included and the project's name.

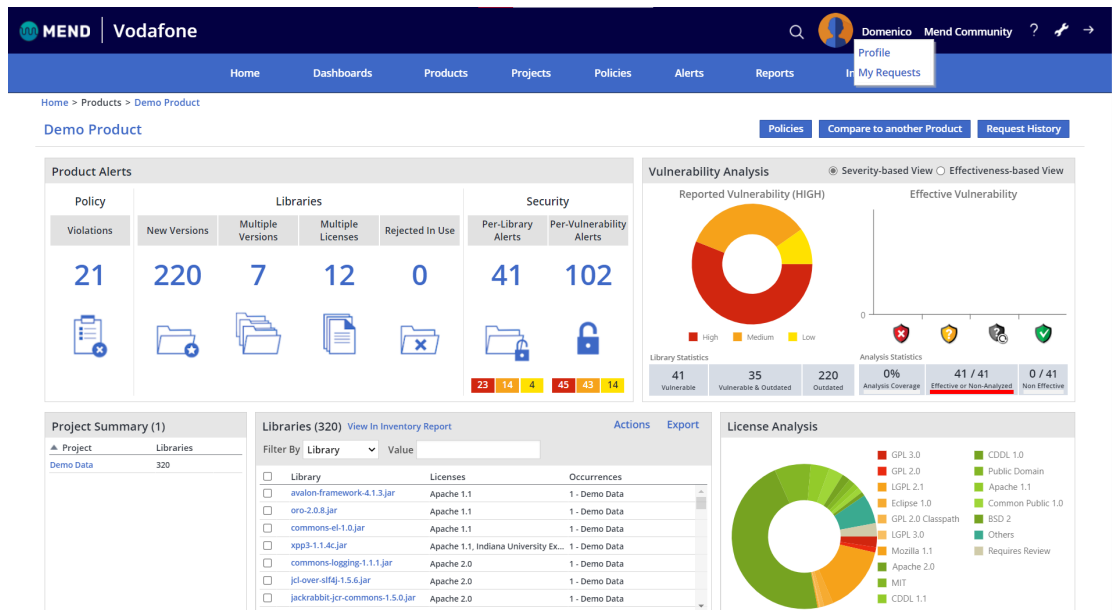


Figure 4.12: Mend SCA product home page

In the Product Alerts tab, at the top-left, we see some numbers referring to the alerts related to the overall product, what we are focusing on are the security alerts, under which we find two measures:

- **Per-Library Alerts:** this number tells how many libraries included in the product present one or more vulnerability;
- **Per-Vulnerability Alerts:** this number tells the overall sum of the vulnerabilities found in the libraries.

These vulnerabilities are divided into three levels of severity, depending on their CVSS Score and following the CVSS v2.0 Ratings system. The CVSS v2.0 Ratings and CVSS v3.0 Ratings classifications of vulnerability severity are reported in the tables below.

Severity	Base Score range
Low	0.0 - 3.9
Medium	4.0 - 6.9
High	7.0 - 10.0

Table 4.5: CVSS v2.0 classification

Severity	Base Score range
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Table 4.6: CVSS v3.0 classification

In the Vulnerability Analysis tab, at the top-right, we see a graphical representation of the number of per-library vulnerabilities of the product and their severity: on the left-hand side we see a pie chart reporting the severity of the found vulnerabilities, with red for representing high severity vulnerabilities, orange for medium severity vulnerabilities and yellow for low severity vulnerabilities. On the right-hand side we find the Effective Vulnerability histogram, which reports the EUA results, if enabled on the given product: the chart reports if the vulnerable functions which have been found in the open-source libraries are actually called and used in the source code of any of the product's projects. The vulnerabilities can be classified in three ways:

- **Effective (red flag)**: vulnerabilities which are actually present in the product's source code and thus need to be prioritized in the planning of remediation actions;
- **Suspected as Effective (yellow flag)**: security alerts which are suspected to be effective in the source code, but need further analysis by the security team;
- **Ineffective (green flag)**: security alerts which are not actually present in the source code and so are not to be prioritized, but still to be fixed in order to avoid them to become effective.

Each of the bars representing effectiveness of the security alerts is then divided into the colors that represent the severity of the vulnerabilities, so to know the amount of vulnerabilities of each type per effectiveness.

If we go into the details of the vulnerability analysis, we can find a page like the one in Figure 4.13.

Here we have a view by library of the security alerts, that is we have a list of all of the libraries used in all of the onboarded products which have been found to have a known vulnerability inside them: for each library we have the total of the overall security alerts and the total vulnerabilities per severity. For example in the figure above we see that the library "spring-core-2.5.jar" used in the product

Security Alerts: View By Library 41 All Time Demo Product All Projects of Demo Product Apply Export

Filter Active Alerts

Library	Product	Project	Severity	Total Alerts	Ignored Alerts	Library Type	Creation Date	Modified Date
<input type="checkbox"/> spring-core-2.5.jar	Demo Product	Demo Data	Medium: 2 Low: 1 details	4		java	17-09-2020	13-12-2021
<input type="checkbox"/> commons-fileupload-1.2.1-...	Demo Product	Demo Data	High: 5 Medium: 1 details	6		java	17-09-2020	17-09-2020
<input type="checkbox"/> spring-core-3.2.0.RELEASE.jar	Demo Product	Demo Data	High: 1 Medium: 3 details	4		java	17-09-2020	13-05-2022
<input type="checkbox"/> wicket-1.4.9.jar	Demo Product	Demo Data	Medium: 3 details	3		java	17-09-2020	17-09-2020
<input type="checkbox"/> commons-httpclient-3.1.jar	Demo Product	Demo Data	Medium: 3 details	1		java	17-09-2020	17-09-2020
<input type="checkbox"/> zookeeper-3.2.jar	Demo Product	Demo Data	High: 1 Medium: 1 details	2		java	17-09-2020	17-09-2020
<input type="checkbox"/> plexus-utils-1.5.15.jar	Demo Product	Demo Data	High: 1 Medium: 2 details	3		java	17-09-2020	17-09-2020
<input type="checkbox"/> spring-web-2.5.jar	Demo Product	Demo Data	High: 1 Medium: 5 details	6		java	17-09-2020	30-11-2021
<input type="checkbox"/> cxf-common-utilities-2.2.7.jar	Demo Product	Demo Data	High: 1 details	1		java	17-09-2020	17-09-2020
<input type="checkbox"/> log4j-1.2.12.jar	Demo Product	Demo Data	High: 6 Low: 1 details	7		java	17-09-2020	18-01-2022
<input type="checkbox"/> commons-collections-3.2.jar	Demo Product	Demo Data	High: 5 details	5		java	17-09-2020	17-09-2020
<input type="checkbox"/> commons-beanutils-1.8.0.jar	Demo Product	Demo Data	High: 2 details	2		java	17-09-2020	17-09-2020
<input type="checkbox"/> commons-collections-3.2.1-...	Demo Product	Demo Data	High: 5 details	5		java	17-09-2020	17-09-2020
<input type="checkbox"/> cxf-api-2.2.7.jar	Demo Product	Demo Data	Medium: 1 Low: 2 details	3		java	17-09-2020	03-12-2021
<input type="checkbox"/> jsch-0.1.38.jar	Demo Product	Demo Data	Medium: 1 details	1		java	17-09-2020	17-09-2020
<input type="checkbox"/> jackrabbit-webdav-1.5.0.jar	Demo Product	Demo Data	Medium: 1 details	1		java	17-09-2020	17-09-2020
<input type="checkbox"/> spring-beans-2.5.jar	Demo Product	Demo Data	High: 2 Low: 1 details	3		java	17-09-2020	31-03-2022
<input type="checkbox"/> cxf-rt-bindings-soap-2.2.7.jar	Demo Product	Demo Data	High: 1 details	1		java	17-09-2020	17-09-2020
<input type="checkbox"/> hudson-core-2.2.0.jar	Demo Product	Demo Data	High: 2 Medium: 1 details	3		java	17-09-2020	17-09-2020
<input type="checkbox"/> hibernate-validator-4.2.0.Final	Demo Product	Demo Data	Medium: 2 details	2		java	17-09-2020	27-06-2021
<input type="checkbox"/> cxf-rt-core-2.2.7.jar	Demo Product	Demo Data	Medium: 1 Low: 3 details	4		java	17-09-2020	17-09-2020
<input type="checkbox"/> libpam4j-1.4.jar	Demo Product	Demo Data	Medium: 1 details	1		java	17-09-2020	17-09-2020
<input type="checkbox"/> groovy-all-1.8.1.jar	Demo Product	Demo Data	High: 1 Medium: 1 details	2		java	17-09-2020	11-11-2020

Figure 4.13: Mend SCA vulnerable libraries list


“Demo Product” under the project “Demo Data” has three medium severity vulnerabilities and one low severity vulnerability, with a total of four alerts for this library.

If we go into the details of one of these libraries, we find the details of the vulnerabilities contained into this library, as in Figure 4.14.





Here we can find an overview about all the vulnerabilities of the library, a short description of the problem and of the suggested remediations. Vulnerabilities are uniquely identified by their Vulnerability Id, which is a code that can either start with CVE, for vulnerabilities identified by the NVD, or WS, for vulnerabilities that have been found by the Mend security research team. Each vulnerability reports both the CVSS v2 and v3 scores, for the sake of completeness.

If we click on one of the vulnerabilities we find the details of that vulnerability, in a page like the one we see in 4.15.

In this Security Vulnerability page we find additional details about the vulnerability, what are the libraries affected by this vulnerability (which could be also libraries other than the one we reached this page from) and the projects in which there are occurrences of the vulnerable libraries, but not necessarily of the particular vulnerable functions. On the top-right panel we see all the details of the CVSS v3 Base Score Metrics, with all of the values that have been assigned to each one of the metrics, both Environmental and Impact.



MEND
Vodafone



Domenico
Mend Community
?



Home
Dashboards
Products
Projects
Policies
Alerts
Reports
Integrate

Home > Security Vulnerability (CVE-2021-22096)

Security Vulnerability


General Details

Name	Severity	CVSS 3 Score	CVSS 2 Score	Date	Modified
CVE-2021-22096	Medium	4.3	4.0	28-10-2021	28-04-2022


In Spring Framework versions 5.3.0 - 5.3.10, 5.2.0 - 5.2.17, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries.

CVSS 3 - Base Score Metrics


Exploitability Metrics

Attack Vector (AV) 


NETWORK

Attack Complexity (AC) 


LOW

Privileges Required (PR) 

LOW


User Interaction (UI) 

NONE


Scope (S) 

UNCHANGED


Impact Metrics

Confidentiality Impact (C) 

NONE

Integrity Impact (I) 

LOW

Availability Impact (A) 

NONE

Vulnerable Libraries

Library Name	Occurrences
spring-webmvc-4.3.13.RELEASE.jar	1
spring-web-4.3.13.RELEASE.jar	1
spring-core-5.0.9.RELEASE.jar	2
spring-web-5.0.9.RELEASE.jar	2
spring-core-4.3.13.RELEASE.jar	1
spring-core-3.2.0.RELEASE.jar	1 - Demo Data
spring-webmvc-5.0.9.RELEASE.jar	2

As previously said, there are many types of AST techniques, each of them with its purpose and target: it follows that there are many more tools that allow performing

such security testing.

In the **DAST** world we find tools such as Netsparker, Acunetix and Indusface WAS: each tool has its characteristics in terms of tested application types, integration and reporting capabilities, and many more.

IAST tools are often provided by companies that already provide SAST and DAST solution, since this is a type of analysis that brings together several features of the static and dynamic analysis. Indeed, among the providers of IAST solutions we find Veracode, HCL (AppScan) and Checkmarx.

IDE plugins are also a very important tool to be integrated in the development process: other than the technical advantages, having a tool that is directly used and seen by the developer that points out the possible security flaws helps him to broaden his security culture and in the future it may be more likely to consider security aspects since the beginning of the project. An example of this, as mentioned above, is the SonarLint plugin by SonarSource: it is a free and open-source plugin and it is available for the most popular IDEs, including Eclipse, IntelliJ IDEA, Visual Studio and VS Code.

SonarLint is a tool that provides instant feedback about bugs and vulnerabilities as code is written, so that the developer is made aware and is able to immediately fix them by following the provided suggestions for remediation. This allows the developer to know about the best coding practices and how to employ them and not produce bad quality code, providing a long-term improvement on the involved people.

4.4 Resources and teams

There are several people and teams to be involved in the DevSecOps process: we must remember that one of the main characteristics of this Agile methodology is to optimize the speed of delivery by creating cross-functional teams, involving people with different backgrounds. We will now identify the types of professional figures and have an overview about what their role is in the company and what part they will play in the DevSecOps process.

4.4.1 Security

The security team plays for sure a central role in the DevSecOps process: while the security culture should be as widespread as possible (for example by adopting a Security Champions model, see later), the security team is in charge of verifying

that the required controls are put in place, that developed software meets the security requirements, that remediation actions, if needed, are prioritized, planned and executed, and many more tasks. It is fundamental that the team works with developers to integrate security tools and practices seamlessly into the existing SDLC.

4.4.2 Developers

Along with the security team, the development team is another pillar of the DevSecOps process: developers are the ones that are in first place in charge of not introducing bugs, flaws and vulnerabilities in the products' source code and to do so they must be adequately trained about the DevSecOps culture. The main issue in the relationship between developers and security is that often the former see the latter as an obstacle in their journey of delivering software in a fast way, the target is to develop and deploy working software, eventually dealing with security issues further in time or when more budget is available: what must be understood is that not preventing security problem may lead to a number of problems in the worst case scenario of a security breach, including lots of additional work to remediate, reputational and economical damage for the company, and so on.

4.4.3 IT Operations

The IT Operations team is in charge of setting up and maintaining the infrastructure upon which the tools and the final developed products and applications will be running: this task has to be performed keeping into consideration the business requirements about service delivery, for all users.

Among the various duties of the ITOps teams we find [28]:

- **Network infrastructure management:** that includes its configuration, monitoring its health and traffic flows and hardware deployment and upgrading;
- **Server management:** provisioning, configuration and management of IT infrastructure for both internal users, such as DevOps teams, and external users, such as customers and partners. This may include the usage of on-site and off-site resources, like data centres and cloud infrastructure resources;
- **Incident and security management:** in the context of the DevSecOps process, this is a crucial task to be carried out in collaboration with the security team. The infrastructure must be designed and implemented in a way that prevents and resists to attacks, through identity and access controls, redundancy, duplication, security protection systems and constant monitoring.

4.4.4 DevOps

The DevOps team has the duty of effectively implement the homonym methodology to develop and deploy software in an Agile way, through Continuous Integration and Continuous Deployment. This team has several tasks whose aim is to reduce distances between development and operations teams, by trying to automate as much as possible the flow between the two environments in a seamless way through proper tools.

Some of the duties of a DevOps team are [29]:

- **CI/CD Pipelines:** proper tools must be selected and integrated to automate as much as possible the build, test and deployment phases, so to improve productivity by not wasting time on lengthy and error-prone activities;
- **Architecture management:** for each product to be developed, a proper architecture must be chosen, designed and deployed by the DevOps architect, being it a microservices, cloud or serverless architecture;
- **Infrastructure as Code (IaC):** this method for managing infrastructure helps reducing the boundaries between development and operations, so it is the DevOps team's job employ the tools needed to adopt this method for further automating provisioning of the infrastructure;
- **Cloud migrations:** in the process of the migration from on-premises to cloud-based applications, the DevOps team is the main player. This process is to be carefully planned, by analysing the technical requirements and choosing a proper cloud provider, up to the design of the infrastructure for the application migration, starting from the least complex apps and finally migrating the data to the cloud;
- **Security compliance:** as all of the systems of the company, the DevOps ones must also be security compliant to the organizational policies. Also CI/CD requires security to be integrated starting from the planning phase in order to have a secure product at the end of each release cycle, and that is the main target of DevSecOps: this integration of security in DevOps environments must also be automated, through integration of AST tools and scripts that check at each delivery that the product is still security compliant and no deviation has been introduced. This task is clearly conducted in collaboration with the security team.

4.4.5 Business owners

Business owners and Product Managers are the ones that dictate the requirements of the product, both functional and non-functional. It is important that among the requirements are also defined some security ones, in the form of user stories that will then be inserted in the Product Backlog and inserted in sprints as every other story referring to other kinds of requirements.

4.4.6 Tribes, Squads and Security Champions

Vodafone has defined its own view of the **Security Champions Model**: in a few words, a Security Champion is a person of the team in charge of developing a project that is not a full-time security expert, but has some knowledge about it and has the duty of assuring that security is integrated in all of the phases of the SDLC.

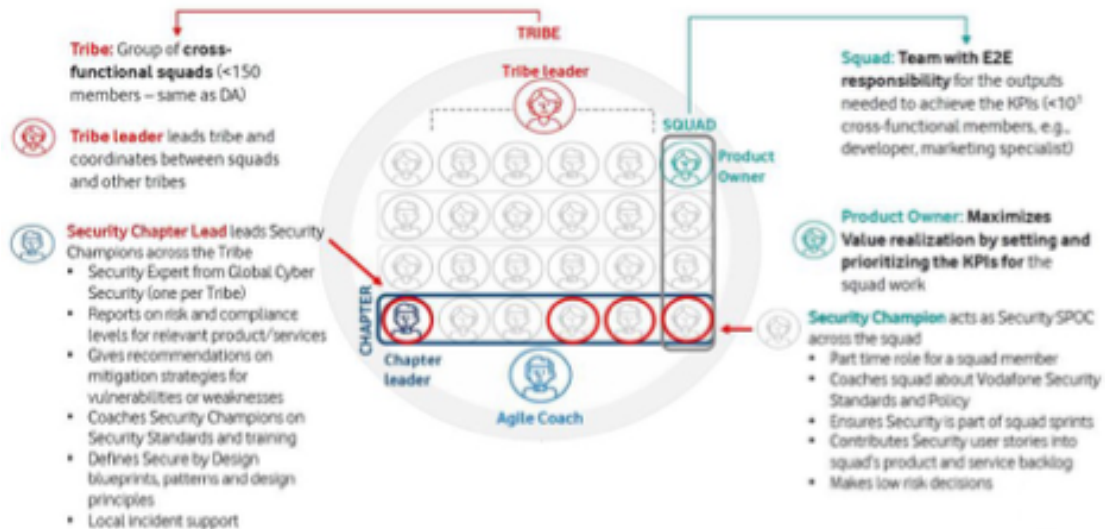


Figure 4.16: Vodafone Tribes, Squads and Chapters model

As we see in Figure 4.16, the top-most entity is the **Tribe**, along with its Tribe leader: a Tribe is a group of cross-functional squads and the Tribe leader is in charge of coordinating and overseeing the work of all the Squads.

The cross-functional team that has end-to-end responsibility for the development of the project is the Squad: a Squad is composed usually by six to ten people, each of them having different skills and giving his/her contribution to the final product. Each Squad may have different figures in it, depending on the type of developed product, but there are two which are mandatory in every team:

- **Product Owner:** it is the one in charge of managing the overall work of the Squad, he/she defines and prioritizes the KPIs to focus and work on at a given moment;
- **Security Champion:** it is a part-time role for one of the Squad members, who has followed adequate training to ensure that security is embedded in the work of the Squad on its project. He/she is the Security SPOC for the Squad.

As we can see in Figure 4.17, each Squad can include different types of professionals in it.

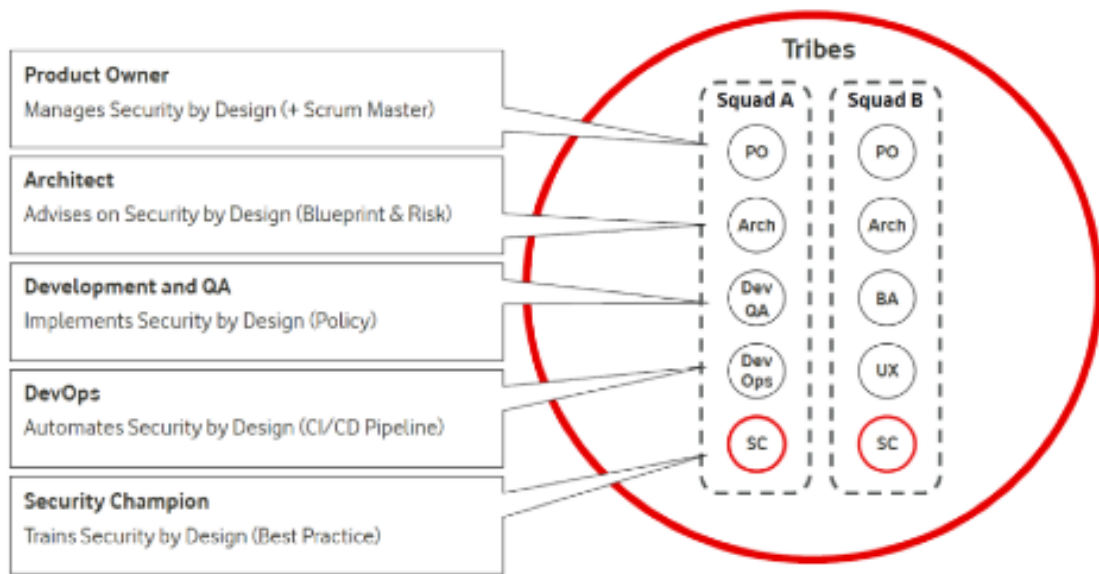


Figure 4.17: Vodafone Squads examples

Security Champions are part of the **Security Chapter** of their Tribe: each Security Chapter has a **Security Chapter Lead**, which, unlike the Security Champion, is a full-time role for a security expert who is part of the Vodafone Global Cyber Security Team and is in charge of:

- coaching the Security Champions about the company's latest security policies and the current best practices to be adopted in project development;
- giving recommendations about the best mitigation actions;
- reporting about risks and compliance about the products that the Tribe works on;
- defining Secure by Design blueprints, patterns and design principles.

This model would ensure that a secure Agile development model is adopted throughout the company to deliver products fast and in a secure way.

4.5 Processes

In the Vodafone definition of the DevSecOps operating model there are two main processes to be considered and integrated in the product development phases: the SDLC and the SPDA.

We have already defined the Software Development Life Cycle as the process that manages the idea, design, implementation and maintenance of the software product. We will later deep dive into the detailed phases that compose it.

The Security and Privacy by Design Assurance (SPDA) process has been defined by Vodafone with the target of assuring that every new product, service and operation meets the company requirements in terms of compliance to security and privacy organizational policies and applicable external laws and regulations. This process is composed by several phases, each of them mapping to one or more phases of the product development, and can be integrated both into Waterfall and Agile development models.

4.5.1 SDLC phases

There are many definitions of the phases that compose the SDLC process, for the purpose of the definition of our DevSecOps operating model we will define a 9-phase process including the following steps:

1. **Idea/Concept:** this is the very starting phase, in which the concept of the new product to develop comes to life and we want to translate it in a concrete solution to be designed, implemented and deployed;
2. **Analysis:** requirements are gathered from business owners and customers, the target of the product and what problem it aims to solve are analysed. Different solutions to the problem may be taken into consideration and costs and benefits of the chosen solution are analysed by performing a feasibility study on several aspects [30]:
 - **Economic:** does the company have the required resources to develop this product?
 - **Legal:** based on the product's scope, is the company able to guarantee compliance to the organizational policies and external regulations?

- **Operational:** can the company satisfy the requirements given the defined operational framework and workflows?
 - **Technical:** what are the available technologies and human resources to support the SDLC process?
 - **Schedule:** can the product be delivered in time?
3. **Requirements analysis and definition:** functional and non-functional requirements are formally written in a Software Requirements Specifications (SRS) document, so that stakeholders and developers can eliminate any remaining ambiguity before starting to work on the development of the product;
 4. **Design:** the product architecture, functions and operations are here defined in detail through screen layouts, business rules, process diagrams, pseudocode and other documentation;
 5. **Development:** in this phase the actual code implementing the defined functions is written;
 6. **Integration, building and testing:** the developed modules are integrated together and undergo a series of functional and non-functional tests in order to see if the product meets the defined requirements and does not contain any bugs or errors;
 7. **Deployment:** the final product is put into production and made available to the final users;
 8. **Operation, maintenance and monitoring:** the working product must be monitored and maintained, if any change is needed or required by the stakeholders;
 9. **Disposal:** in this final phase work is planned to dismiss the current system and transition to a new one, by replacing the proper hardware and software and archiving or destroying information depending on what are the privacy requirements. This is a very sensitive step because it must be sure that no obsolete system is left available for malicious users to exploit them and eventually disclose information in an unauthorized way.

4.5.2 SPDA phases

The SPDA process is divided into six phases, each of them with a defined target. The phases are:

1. **Idea/Concept:** target is to identify and assess security and privacy risks so that proper controls are designed into the product and requirements that cannot comply with the organization's policies are eliminated prior to dedicating time and resources to them;
2. **Design:** a risk assessment and control identification must be performed to ensure that the final design incorporates the defined security controls to be implemented;
3. **Build:** the security controls that were identified in the first phase and designed in the second are actually built, implemented and integrated into the product;
4. **Test and Sign-Off for Launch:** in this phase, that must be carried out during or at most at the end of the build phase, it must be verified that the designed security controls have been implemented and they comply with Vodafone security and privacy requirements and external applicable laws and regulations;
5. **In-life:** in the case that new features or significant updates are introduced into the product, the steps above must be repeated in order to ensure that the product still meets the security and privacy requirements after the introduction of the updates;
6. **Decommissioning:** the security and privacy requirements have to be addressed also at the end of life of the product, by properly decommissioning the used hardware and software and disposing data as required by the regulations

Chapter 5

DevSecOps Operating Model

In this chapter we will go into the details of the new DevSecOps Operating Model by analysing the nine phases of the DevSecOps Continuum: for each one we will define the involved people, processes and technologies and how they will interact, with defined roles and responsibilities.

5.1 Overview

As the Agile methodology requires, the main idea is to have a **cross-functional team** composed by several types of figures, each with its own set of skills that will give a significant contribution to the development of the product: this, as previously defined, will be a Squad, a group of people that will have end-to-end responsibility for the new product, service or feature developed. Depending on the **type of product**, proper professional figures will be identified and included in the Squad.

In the DevSecOps Operating Model each phase of the Continuum will be mapped to the corresponding supporting processes, which are the SDLC and the SPDA presented in the previous chapter, and technologies.

5.2 Plan

People

In this starting phase of the product lifecycle, everyone is involved in the definition of the product characteristics, features and requirements, from the Product Owner and stakeholders up to the developers that will implement the product. Of course the Security and Privacy teams play a crucial role since the beginning, also by employing the Security Champions.

Processes

The involved phases of the SDLC and SPDA processes at this stage of the model are the ones that relate to the definition of the required product, starting from the customer's need to be satisfied and then coming up with the most suitable solution for fulfilling these necessities: it is in this moment that the functional and non-functional requirements are gathered from all of the involved stakeholders, they are analysed and their feasibility is ensured. Finally the most proper architecture for the product is selected among various possibilities and we define the needed components and how they interact to offer their services.

The phases of the SDLC process which are then involved in this step are **Idea/Concept** (1), **Analysis** (2), **Requirements analysis and definition** (3) and **Design** (4).

The results of the SDLC process must be assessed also from the security point of view during the first two phases of the SPDA process: **Idea/Concept** and **Design**. In the first phase the Security and Privacy teams receive as an input from the Business team the concept of the product, for example in the form of a High Level Design (HLD): given the provided information, an analysis is conducted to identify the risks that are inherent to the product and, if the risks can be mitigated, the controls that are needed to do so, otherwise the remaining risks need to be assessed and understand if they can tolerable or not. Also, it is important to identify possible flaws in the design to be eliminated prior to dedicating resources to their development: all of this can be done by employing proper Threat Modelling techniques.

After the controls to be implemented are found and communicated to the Business team, a go/no-go decision is made based on the required resources and effort. If the project continues, then the result of the Design phase must be assessed in order to ensure that the possible flaws that were previously identified have not been included in the design and instead the defined controls have been properly inserted into the design documents.

Technologies

During the Plan phase there are no particular technologies that are necessary in order to produce the **requirements and design documents**, of course there are tools that help in having better results in an efficient way: for example **planning and issue tracking tools** such as the Jira suite can help a lot in the definition of the tasks and people assigned to them, the planning of the sprints and tracking of the opened issues, to check whether they are still to be reviewed or someone is

already working on it or if it has already been solved and closed.

In order to store and manage the produced documents, containing the requirements, user stories, design, etc., a **VCS**, such as Git, could be useful to track their creation and changes, when they are performed and who did them, so it is easier to make everyone that is involved responsible and accountable for what they produce.

Roles and Responsibilities

As stated by the SPDA process, the Security and Privacy teams will be in charge of ensuring that no requirement that cannot comply with security policies is introduced and that the proper security controls are included based on the remaining requirements.

The Security Champion of the Squad will also be in charge of including security requirements in the defined documents, including security user stories.

5.3 Code

People

This is the phase in which the required functionalities and the components defined in the architecture are implemented and coded by the developers, who must be supported by the security team which supports the coding stage by providing training on secure coding practices and about the usage of the proper code analysis tools and interpretation of their results.

Processes

The phases of the processes that are involved at this stage are the ones that are related to the development of the components that were defined in the design documents.

For the SDLC process we have the **Development** (5) phase, while for the SPDA process we have the **Build** stage, in which the security controls which were identified in the previous stages are integrated into the developed component. If there are changes during the Development phase with respect to what was defined in the Design phase, a small iteration of the Design and Build phases of the SPDA process must be repeated in order to ensure that the proper controls are still in place and no further risk has been introduced.

Technologies

The required technologies in this stage are the ones enabling the developers to write code and build the product components: these tools can be as simple as a text editor, but in order to write complex programs more sophisticated tools must be used, such as **IDEs** and necessary plugins. Depending on the type of developed product, a particular IDE may be more suitable than another or it may even be the case that the choice is forced by some requirement or design constraint: for example, for developing a mobile application an IDE such as Android Studio would be an excellent choice, but if we want to make an iOS version available we may be forced to use the Xcode IDE.

The most used IDEs and code editors also provide a very large set of plugins which can support almost any kind of necessity by the developer: in this Operating Model two types of plugins that would be necessary are those that allow easy integration with VCS and those that provide real-time code analysis to the developer. In the previous chapter we mentioned SonarLint, the IDE plugin provided by SonarSource that performs exactly this task: it helps the developer to spot possible bugs and vulnerabilities as they are coded, with the benefits that the developer immediately realizes and fixes the problem and learns a lesson that will help him/her in the future to hopefully not introduce any more flaws of that kind.

Code repositories and **VCSs** are also essential tools in this phase: the code written by the developers is pushed onto the repositories, which may be hosted on platforms like GitHub, GitLab or BitBucket if the chosen VCS is Git. Other developers in the team can then see changes to the code just by pulling the latest version on their local machine: each change is tracked both in terms of when it was performed and who performed it.

Roles and Responsibilities

In the Code stage the main responsibilities are on the Development team, which is in charge of implementing the defined components by writing custom code and leveraging on any needed third-party component or library. In the meanwhile, the Security team must monitor the progress of the work, for example by checking that code analysis tools do not report any critical vulnerability on the code which may be pushed on a daily basis: if that happens, the developers must be alerted about the fact that there is some code that needs to be fixed and assigned a deadline depending on the severity of the vulnerability.

Another task of the Security team is to support the developers in the adoption of the secure coding practices and the usage of the code security tools: policies

and procedures must not be “thrown over” to the developers, the introduction of security inside the coding stage must be a gradual process, otherwise developers will just continue to see security as an additional step that delays the deliver of “real” functionalities, which is what they want to achieve. This support will be also given by setting up dedicated training sessions about secure coding, usage of code review tools and how to interpret their outputs.

When developers need to rely on third-party components, which is a case that happens almost always in software development companies, the usage of those components must be subject to approval of the Security team, which must assess that the component is compliant to the company’s requirements and can be safely used for developing new products.

5.4 Build

People

In the Build phase the components that have been coded by the developers and the third-party ones are built together to create the final system to be delivered and deployed. Here also the DevOps team must be involved, in order to set up the tools and the infrastructure needed to automate the build pipeline. The Security team is also involved for analysing and suggesting possible security tools to be integrated in the build pipelines.

Processes

The phases of the processes that are involved in this stage of the model are the ones related to the build and integration of the system, which are the **Integration, Building and Testing** (6) phase for the SDLC process and the **Build** stage for the SPDA process.

This is the stage in which the build of the application is run and all of the modules are integrated together: here further checks are performed on whether the defined security controls have been implemented and no modification that can potentially introduce further risks have been performed.

Technologies

There are several types of tools that can help in this stage: **build automation tools** and **CI/CD servers** plus **SAST** and **SCA** tools for introducing security analyses.

Build automation tools, such as Maven, are very useful for the compilation and linking of source code files into binary code, packaging the produced executables and, possibly, running automated tests.

CI/CD servers, such as Jenkins, are used to implement Continuous Integration features: their role is to get the code which is pushed into the repositories and put it through a pipeline composed by several steps defined by its programmer. Build can be a part of the process implemented by this tool, in fact Maven could be integrated as a build automation tool inside a CI/CD pipeline defined in Jenkins. These tools are customizable through lots of plugins that perform additional steps which may be required. The runs of the defined pipelines can be scheduled to be executed at particular times or triggered each time new code is pushed into the repositories: these runs could take quite a long time depending on the provided steps, so care must be taken in choosing when to run the pipelines in order to not slow down the process, instead of speeding it up, which is the main advantage of using these tools.

From the security point of view, this is the stage in which tools for performing SAST and SCA analyses can be performed: these can be either integrated in the build pipelines or run manually on the available code, using tools such as SonarQube and Mend SCA.

Roles and Responsibilities

In this phase the main role is the one of the DevOps team, which has the duty of setting up the necessary tools and configure the build pipelines and possibly fix any issue related to them.

The role of the developers is to push the code they have written into the dedicated repositories, so that the CI process can take place.

The Security team in the meanwhile is in charge of monitoring the security aspects of the build process: they must analyse and suggest proper tools to analyse the pushed code during this phase, which means implementing those security tests that can be performed on source code, namely SAST and SCA. For implementing these tools, collaboration with the DevOps team is required.

After having successfully implemented those tools in the pipelines, the results of their analyses must be monitored through proper dashboards: if any critical issues are found, such as flaws introduced in the source code or vulnerable components used inside of the product, the Development team must be notified, possibly in

an automated way. Following that, remediation action must be performed by the developers, such as fixing some given lines of code or updating or replacing a vulnerable component: this should always be done with the support of the Security team, at least until the developers are properly trained on the security issues and how to solve them.

5.5 Test

People

This is the phase in which the artifacts produced by the developers are taken by the testers and QA teams to test the product to be delivered against the defined requirements and verify that it is ready for releasing. The DevOps and Security teams are still involved in this phase to integrate tools in the CI/CD pipelines and check the product compliance against the security policies.

Processes

The Test stage of the DevSecOps Continuum is still mapped to the **Integration, Building and Testing** (6) phase of the SDLC process: here the build artifact obtained by integrating all of the components undergoes various tests in a proper testing environment, which is usually made as similar as possible to the production environment. The tests are performed by a dedicated QA team, which then reports the results of the tests and state if the product meets the functional and non-functional requirements and delivers its functionalities without any problem.

Regarding the SPDA process, we enter in the **Test and Sign-Off for Launch** phase: this is the phase in which the Security and Privacy teams verify that the defined security controls have been implemented and that the product is compliant to the company's security policies. During this stage, dynamic analyses are conducted on the product in execution and the environment in which it is deployed.

Technologies

The tools employed in this phase are the ones allowing to run various kinds of tests: unit, integration, system-level tests and also those types of tests that are applicable only to particular types of products, such as tests on the graphical interfaces of web and mobile applications.

Regarding the security point of view, here we use those tools aiming at assessing the product security at runtime, that is to say **DAST** and **IAST** tools. If a

deeper security analysis is needed, then a complete pentest can be run on the given product, but that is a more time-consuming activity that requires some preliminary analysis, tests execution and results reporting.

Roles and Responsibilities

During this phase a central role is played by the QA team, which is the one that, given the requirements of the product, test it in order to check if it meets them or if it needs to be sent back to the Development team in order to fix some bug or improve some aspect of the user experience.

To the results provided by the QA team, the Security team must add those related to the compliance of the product to the security requirements of the company and of the product itself: the two teams must work together in order to identify what are the possible remediation actions to be suggested so that both the usability and the security of the application are improved.

5.6 Release and Deploy

People

After the QA and Security teams assess that the product meets the criteria to be published, its release is planned, and the DevOps and Operations team take care of the setup of the infrastructure that will be dedicated to host the application and offer its services to the customers.

Processes

These stages correspond to the **Deployment** (7) phase of the SDLC process, the one in which the infrastructure for hosting the application is set up and the product is finally deployed in the production environment and made available to the customers.

On the SPDA process side, we are still in the **Test and Sign-Off for Launch** phase: based on the results coming from the tests, a go/no-go decision is made about the publication of the product and if this decision has a positive outcome, the product is deployed.

Technologies

The technologies to be adopted in this phase depend on the type of infrastructure that will be adopted to deploy the product: in case of an on-premises infrastructure, the most proper tools to be employed are those allowing **automatic configuration**

of the involved systems, such as Puppet, Chef or Ansible, which was presented in the previous chapter.

In case of a cloud-based infrastructure, since we do not have direct access to the machines that will make up our system, cloud providers allow configuration through **IaC**: the setup of our infrastructure will be performed through particular scripts that describe the desired state usually through some DSL.

In case that the chosen deployment model involves the use of containers, a tool like **Docker** could be very useful to create and deploy the product in a containerized form.

Roles and Responsibilities

During this phase, the Security team must work along with the Operations and DevOps teams to ensure that the infrastructure is constantly updated, properly configured and monitored: many times attackers exploit vulnerabilities that are found in out-of-date components that can be resolved by just updating the affected component. Also bad configurations of the infrastructure can be easily exploited by malicious users: a big advantage of the IaC approach is that the scripts that are used for configuring the infrastructure are more easily checkable for flaws in the described configuration, instead of the manual configuration procedures that were once needed to properly set up the infrastructure. A problem of this kind could mean significant issues in terms of data leaks, which may be also sensitive data, thus causing privacy-related problems that can lead to economical and reputational damages for the company and even the opening of legal disputes.

5.7 Monitor and Respond

People

In these phases the product is in operation in the production environment and metrics related to its performance must be identified and collected: based on the monitoring results, it may be necessary to respond to issues and incidents, this is why in these phases the Security and Operations teams need to collaborate to promptly identify and resolve any kind of problem that may arise in the operation of the product.

Processes

These stages are related to the **Operation, Maintenance and Monitoring** (8) phase of the SDLC process, the one in which the product is in operation and

providing its services to the customers: during this phase the product and the infrastructure it runs on must be continuously monitored and checked for any anomaly or suspicious behaviour to be further investigated. Also, analytics about the user behaviour and experience can be collected to provide some metrics about the product's trend.

Regarding the SPDA process, we enter in the **In-life** phase: during this phase, if any new feature is to be added or significant update is needed to be performed on the product, the preceding phases of the SPDA process must be repeated in order to ensure that the changes do not introduce any new risk that is not properly managed.

Technologies

The tools that can be used during these stages are those for monitoring and maintaining the health of the services and of the infrastructure they run on: in case of containerized applications, a tool like **Kubernetes** could be used, thanks to its orchestration features that allow having automatic allocation of resources and containers to the applications, load balancing among the different containers and self-healing capabilities in case of containers freezing or not working correctly.

Under the security point of view a very recent and innovative solution consists in the usage of **RASP** tools, which are based both on data coming from the executing application and from the external world in order to detect and block possible attacks.

Roles and Responsibilities

In these phases the Security and Operations teams must collaborate in the monitoring of the behaviour of the application and the services it offers: whenever an unexpected event is detected, the teams must be alerted and the issue has to be analysed. If the analysis results in the need for a component to be replaced or updated related to the infrastructure, the Operations team must perform this task, always under the supervision and approval of the Security team. The teams must also be ready to respond to a possible security incident, which, even if one of the main targets of this model is to avoid them, are never completely excluded from happening.

5.8 Improve

People

This is the phase that restarts the DevSecOps cycle: based on the metrics and feedback collected in the previous phases, during the operation of the product, any necessary fix or improvement is planned and designed, exactly as it was done during the planning phase and involving the same people.

Processes

This stage concludes the cycle by going back to the beginning: after improvements have been identified, they must in fact be planned and designed exactly as the starting product was defined. The SDLC process phases mapped to this stage are then the **Analysis** (2), **Requirements analysis and definition** (3) and **Design** (4) phases.

From the SPDA point of view we are still in the **In-life** phase, because it is the one that manages the steps to be performed when a new feature or update has to be introduced in the application and it needs to be ensured that it still meets the security and privacy requirements.

Technologies

As this stage is very **similar to the Plan** stage, the tools that support it will be the same ones, being them tools for creating requirements and design documents, repositories in which they are stored and tracked and planning and issue tracking platforms, such as Jira.

Roles and Responsibilities

In this stage all of the parties that were involved in the definition of the initial product requirements are again involved to take some decisions about the improvements to be done on the product: based on the analytics collected during the product operation, the stakeholders may want to perform some modifications that must then be discussed with the Development, Security and Privacy teams in order to ensure that these updates are feasible both in terms of required functionalities and in terms of security compliance.

5.9 Decommissioning

Even if it is not properly a part of the DevSecOps Continuum, decommissioning is a part of software lifecycle. Every system will eventually come to a point in which the benefits that come from the functionalities it provides will be overcome by the problems in its maintenance, which can be caused by several factors, such as the size of the project that may become too large to be properly maintained or the components that make the product up becoming legacy and not anymore supported by their producer: these may be components of strategic importance inside of the product which cannot be substituted without major refactoring of the product. This is usually something that the companies do not want to deal with or cannot afford in terms of time and resources.

The consequence of these problems is that the product comes at its end-of-life stage and it must be substituted by a new one that still provides the functionalities of the previous one, but using components which are either maintainable by the developers or supported by the third-party provider they come from.

Particular care must be taken in the decommissioning of end-of-life products: it must be ensured that no obsolete system upon which the product was running is left available for exploitation from attackers. Usually those systems, given the fact that they are not useful anymore for providing a service, are ignored and not updated: this makes them a perfect entry point for attackers which could then have access to systems that are actually in operation and cause severe damage.

Also data that was stored and processed by these end-of-life system must be properly treated in the ways that are stated by the regulations: depending on the type of data, it may be needed to archive or even destroy them.

These subjects must be treated by the **Security, Privacy and Operations** teams in collaboration and there are dedicated phases of the SDLC and SPDA processes that deal with this step, namely the **Disposal** (9) and **Decommissioning** phases.

Chapter 6

Conclusions

In this final chapter we are going to draw some final considerations about the performed work, what its outcomes have been and what are some possible future developments to be performed to fully deploy a DevSecOps strategy.

6.1 General conclusions

This thesis work has been carried out with the Cyber Security team of Vodafone Italia and it lasted for six months, from December 2021 to June 2022. During this time, also with the support of the global Cyber Security community, an intensive study of the company organization, strategy and security policies and procedures has been performed. Then, the Agile methodology and the principles of DevSecOps have been analysed to understand their key points and the measures to be taken in order to be effectively implemented.

This thesis work's starting point was a set of technologies, processes and people which were already in place for employing a DevOps software development methodology based on an on-premises infrastructure: the final aim of the work was to identify a model in which all of the above mentioned, plus any further additional resource that could be needed, could be integrated in order to deploy an effective DevSecOps strategy for secure Agile software development on a cloud-based infrastructure.

This was a fundamental step in the process of the transition of Vodafone from being a Telco company to a Tech company: adopting a DevSecOps software development methodology will help in keeping up with the pace of the software and in general of the IT market, where new features and functionalities are developed and deployed every day and a company which wants to be competitive must be able to

quickly adapt to the required changes. In the same way, new security threats are always behind the corner and the products that are offered to the market must be resilient to such threats, in order to not lose the customers' loyalty and not incur in any legal issues, especially if we are talking about products and services which are offered to millions of users and store and process potentially sensitive data.

However, it must be considered that the work carried out in these months is just a part of a multi-year strategic program which has just been started to be deployed, its implementation will carry on for the next few years.

At the moment, following the definition of the DevSecOps Operating Model, the first tools to be integrated have been identified and studied, with the first training sessions delivered to the teams involved in the process. The first Squads are also to be defined and composed for the next projects to be developed following the Agile model, especially including Security Champions that will have to attend dedicated sessions to be constantly updated about the security threats and the company policies and procedures to be protected against them.

6.2 Future developments

Being this work part of a company strategy that will be carried on for the next years, there are of course some future developments planned for the implementation of the DevSecOps model:

- The model must be **widely deployed** on a company scale, after a probing phase, this will require larger efforts in terms of organization of Squads and of proper training sessions about the model and the employed tools;
- Some **metrics** to measure how the model is performing are to be identified and, once the process has been properly deployed, measurements and feedbacks must be collected and analysed in order to identify possible critical points and problems and how to solve them or in general how to improve the process's performance;
- The Security team has to constantly analyse and be **updated about the security threats** and vulnerabilities landscape and must be able to suggest proper prevention measures through the implementation and integration of **new security tools** and new ways to automate them.

Bibliography

- [1] David Chappell et al. «What is application lifecycle management». In: *Chappell & Associates* (2008) (cit. on p. 8).
- [2] Eray Tüzün, Bedir Tekinerdogan, Yagup Macit, and Kürşat İnce. «Adopting integrated application lifecycle management within a large-scale software company: An action research approach». In: *Journal of Systems and Software* 149 (2019), pp. 63–82 (cit. on pp. 9, 12).
- [3] Carol Donnelly. *7 Critical Benefits Of Application Lifecycle Management*. URL: <https://www.entranceconsulting.com/7-critical-benefits-of-application-lifecycle-management/> (cit. on p. 12).
- [4] *DevOps*. URL: <https://en.wikipedia.org/wiki/DevOps> (cit. on p. 13).
- [5] *What is Continuous Integration?* URL: <https://www.jetbrains.com/teamcity/ci-cd-guide/continuous-integration/> (cit. on p. 14).
- [6] *What is Continuous Deployment?* URL: <https://www.jetbrains.com/teamcity/ci-cd-guide/continuous-deployment/> (cit. on p. 15).
- [7] *What are microservices?* URL: <https://microservices.io/> (cit. on p. 16).
- [8] *Microservices*. URL: <https://en.wikipedia.org/wiki/Microservices> (cit. on p. 16).
- [9] *Infrastructure as Code*. URL: <https://www.hpe.com/it/it/what-is/infrastructure-as-code.html> (cit. on p. 18).
- [10] Rileena Sanyal. *What is Continuous Monitoring in DevOps?* URL: <https://www.headspin.io/blog/what-is-continuous-monitoring-in-devops> (cit. on p. 19).
- [11] Jakob Pennington. *The Eight Phases of a DevOps Pipeline*. URL: <https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba> (cit. on p. 20).
- [12] Larry Maccherone. *The DevSecOps Manifesto*. URL: <https://medium.com/continuous-agile/the-devsecops-manifesto-94579e0eb716> (cit. on p. 23).

- [13] Parveen Bhandari. *What is DevSecOps? | The Ultimate Guide*. URL: <https://www.xenonstack.com/insights/what-is-devsecops> (cit. on p. 24).
- [14] *Apply Shift-Left Testing Approach to Continuous Testing*. URL: <https://katalon.com/resources-center/blog/shift-left-testing-approach> (cit. on p. 24).
- [15] *People, Process, Technology: The PPT Framework, Explained*. URL: <https://www.plutora.com/people-process-technology-ppt-framework-explained/> (cit. on pp. 26, 43).
- [16] Roger S Pressman. *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005 (cit. on p. 31).
- [17] *What is IaaS?* URL: <https://www.redhat.com/en/topics/cloud-computing/what-is-iaas> (cit. on p. 32).
- [18] *Secure by design*. URL: https://en.wikipedia.org/wiki/Secure_by_design (cit. on p. 34).
- [19] Eric Raymond. «The cathedral and the bazaar». In: *Knowledge, Technology & Policy* 12.3 (1999), pp. 23–49 (cit. on p. 35).
- [20] *Gartner Says Worldwide IaaS Public Cloud Services Market Grew 41.4% in 2021*. URL: <https://www.gartner.com/en/newsroom/press-releases/2022-06-02-gartner-says-worldwide-iaas-public-cloud-services-market-grew-41-percent-in-2021> (cit. on p. 35).
- [21] *Operating Model*. URL: <https://www.gartner.com/en/information-technology/glossary/operating-model> (cit. on p. 41).
- [22] Andrew Campbell. *What is an operating model?* URL: <https://opexsociety.org/body-of-knowledge/operating-model/> (cit. on p. 42).
- [23] *2022 Data Breach Investigations Report*. URL: <https://www.verizon.com/business/resources/reports/dbir/> (cit. on p. 43).
- [24] T. Scanlon. *Decision-Making Factors for Selecting Application Security Testing Tools*. Carnegie Mellon University's Software Engineering Institute Blog. Aug. 2018. [Online]. URL: <http://insights.sei.cmu.edu/blog/decision-making-factors-for-selecting-application-security-testing-tools/> (cit. on p. 54).
- [25] *SonarQube Documentation*. URL: <https://docs.sonarqube.org/latest/> (cit. on p. 73).
- [26] *Software Supply Chain Attacks Tripled in 2021: Study*. URL: <https://www.securityweek.com/software-supply-chain-attacks-tripled-2021-study> (cit. on p. 77).

- [27] *Mend Software Bill of Materials*. URL: <https://www.mend.io/wp-content/media/2021/11/Mend-Software-Bill-of-Materials.pdf> (cit. on p. 78).
- [28] Muhammad Raza. *IT Operations (ITOps) Roles Responsibilities*. URL: <https://www.bmc.com/blogs/it-operations-roles/> (cit. on p. 84).
- [29] Alfonso Valdes. *DevOps Team: Roles and Responsibilities for 2022*. URL: <https://www.clickittech.com/devops/devops-team/> (cit. on p. 85).
- [30] Muhammad Raza. *The Software Development Lifecycle (SDLC): An Introduction*. URL: <https://www.bmc.com/blogs/sdlc-software-development-lifecycle/> (cit. on p. 88).