POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Master degree in Mechatronic Engineering

Master Degree Thesis

Analysis and proposal of methods to increase the reliability of CNH Industrial vehicle applications

Supervisor:

Prof. Giorgio BRUNO

Candidate:

Hristina ILIEVA

Company tutor:

Ing. Raimundo Marcio PONTES (CNH Industrial)

Academic Year 2019/2020

Abstract

With the continuous advancements in the automotive industry, there is an ever-increasing need to ensure the reliability of the vehicles.

This Master's degree thesis focuses to find a methodology for improving the software reliability in the electronic control units utilized into the CNH Industrial's vehicles. It is a global company formed by 12 brands, including IVECO S.p.A, an Italian industrial vehicle manufacturing company based in Turin.

Starting with the definition of error, fault, and failure, the thesis continues with the summary of software development lifecycle models. In the further chapters, an overview of the verification and validation model has been provided, followed by a description of Agile methodology, DevOps methodology, and Model-based development. The description of each method is needed to propose a final methodology, a combination of Agile + Scrum + DevOps + MBD. It is a suitable methodology that answers the requirements of the company, verifies and validates the control units, and the integration between them from the early beginning until the insertion into the vehicle. Moreover, the adoption of this methodology is expected to increase the overall reliability of the vehicular applications of CNH Industrial.

Keywords: reliability, agile software development, model-based design, Scrum, DevOps, vehicular applications

Sommario

Con i continui progressi nel settore automobilistico, si forma una crescente necessità di garantire l'affidabilità dei veicoli.

Questa tesi di laurea magistrale si concentra sulla ricerca di una metodologia per migliorare l'affidabilità del software nelle centraline elettroniche utilizzate nei veicoli di CNH Industrial. Essa è una società globale formata da 12 marchi, tra cui IVECO S.p.A, un'azienda italiana produttrice di veicoli industriali con sede a Torino.

A partire dalla definizione di errore, guasto e failure, la tesi prosegue con il riepilogo dei modelli del ciclo di vita dello sviluppo software. Nei capitoli successivi è fornita una panoramica del modello di verifica e convalida, seguita dalla descrizione dei metodi Agile, Scrum, DevOps e di sviluppo model-based (MBD). La descrizione di ciascun metodo è necessaria per proporre una metodologia finale, una combinazione di Agile + Scrum + DevOps + MBD. Questa metodologia si propone di migliorare l'affidabilità delle applicazioni veicolari di CNH Industrial mediante la verifica e l'integrazione delle centraline dall'inizio fino all'inserimento nel veicolo.

Parole chiavi: reliability, agile software development, model-based design, Scrum, DevOps, vehicular applications

"And, when you
want something, the entire
universe conspires in helping
you to achieve it".

~Paulo Coelho*, The Alchemist*

Contents

Chapter 1 – Introduction and company overview

1.1. Introduction

The vehicles are of fundamental importance for the people. A person cannot imagine his/her life without using his vehicle, using public transport or using some other type of vehicle for transportation. For some people, the vehicle is even their office, their working place. In the last years, the technology is improving rapidly; the industry is growing enormously fast. The vehicles are following that trend of development in the industry; therefore they are getting better, more innovative and nearly autonomous. One of the main improvements, compared with the vehicle 30-40 years ago, along with the sensors, is the introduction of the software inside the vehicles that control the various sensors, circuits, commands. As the vehicles are developing and improving, the software is improving, too, making the vehicle more sophisticated and modern. Having all of this in mind, it is essential to have a secure vehicle to bring us safely to the destination point. And to design the vehicle to be more secure means to make the software inside it reliable, since the software controls the main functions in the vehicle. Therefore, the preferred way to increase the reliability is to test the vehicle and the software jointly as an entity, until a satisfying result is obtained.

The automotive industry is putting a great effort to sell their products to the clients and to gain their trust. For the companies, the most effective strategy to gain the trust of their customers is to offer reliable products, products that guarantee the quality, security and safety. This Master's degree thesis focuses to propose a suitable method for increasing the reliability in CNH Industrial vehicles. The objective is to analyse the electronic modules, the interaction among them, the possible errors and faults that occur. Additionally, the thesis aims to reach the final point by research and analysis of existing models for

verification and validation, the comparison between them and highlighting their advantages.

Since this master thesis was carried out in the CNH Industrial (IVECO offices) in Turin, the introduction part continues with a brief description of the company and their activity.

Chapter 2 is explaining the distinction between the most common terminologies in software engineering.

Chapter 3 is dedicated to a description of the various software development lifecycle models and the phases that they are composed of.

Chapter 4 deals with the verification and validation process, the difference between the two parts of this process and the methods on how to perform each of them. Moreover, the different types of testing are introduced with their advantages and disadvantages.

In chapter 5 the focus is on the Agile method, an innovative method based on continuous interaction with the stakeholders. The same chapter, in the following section, observes the Scrum framework and the Sprint concept.

Chapter 6 is about the DevOps methodology, which mainly focuses on the communication between the two sectors: development and operations.

An overview of model-based development is given in chapter 7; moreover, a methodology that is expected to increase the reliability of CNH Industrial vehicles is also proposed.

1.2. CNH IVECO

CNH Industrial is a global company that combines the activities of design, manufacturing, distribution and finance of different vehicles. Their span of vehicles goes from trucks and busses to combines, tractors and other agricultural vehicles, but the defence vehicles as well. The company also produces powertrain solutions for both on-road and off-road vehicles. CNH

Industrial (Figure 1 [2]) is formed of 12 brands: CASE Agriculture; STEYR Traktoren; CASE Construction; New Holland Agriculture; New Holland Construction; IVECO; IVECO Astra; IVECO Bus; Heuliez Bus; Magirus; IVECO Defence vehicles and FPT Industrial [1].

Figure 1: CNH Industrial Logo

IVECO is founded in Turin, Italy in 1975. The main activity is the design and production of light, medium and heavy commercial vehicles. IVECO Daily represents the light range, and the vehicles can be vans or minibuses, in the medium range are included smaller trucks with a span from 7t up to 19t known as Eurocargo, while in the range of the heavy vehicles are Stralis, S-Way and Trakker [3].

The goal of IVECO is to offer its customers a safe, secure and reliable vehicle that is also cost-efficient and sustainable. IVECO is a leader from the ecological point of view since diesel and natural gas engines are introduced to all the ranges of the vehicles.

The CNH Industrial is following the trends in the technology and the industry, so the vehicles that are produced are becoming modernized, innovative and improved. IVECO, being part of the 12 brands of CNH does not lag behind in the technological development of its vehicles. IVECO vehicles, from different ranges, won many awards for innovation, design, sustainability in the previous years. In the last years, the direction, in which the progress is

going, is making the vehicles more and more automated, so in the 2016 IVECO deployed semi-automated trucks, which is a big step for the medium and heavy range of vehicles. And it is not stopping there; it is always looking forward to future improvements [1].

Some of the vehicles are represented in Figure 2 [1], one of each range.

Figure 2: Daily (up left), Eurocargo (up right) and Stralis (down)

Chapter 2 – Vocabulary

2.1. Glossary: error, fault and failure

Software engineering is the systematic, disciplined approach to the development, management, operation and maintenance of the software [4]. Which means it is the application of the engineering to the software. During the study of software engineering, or the testing of the software, one can encounter terms that seem synonyms, even though there is a significant difference in their meaning. Such terms are error, fault, and failure. Their definitions can be different from a book to book, but as a reference in this thesis was used The Institute of Electrical and Electronics Engineering (IEEE) Standard Glossary of Software Engineering Terminology.

2.1.1. Error

Error stands for human action that produces an incorrect result. It is usually a misunderstanding, misinterpretation or confusion by the developer. It can be a result of a misunderstanding in the design specification, wrong translation, wrong measurement unit, wrong variable name or other causes. The error in the program leads to a fault in the program itself. Errors are detected when some part of the computer software demonstrates undesired state and can be classified in different categories such as the following:

- Fatal error – an error that results in the complete inability of a system or component to function;
- Dynamic error – an error that is dependent on the time-varying nature of the input;
- Static error – an error that is independent on the time-varying nature of the input;

- Semantic error – an error resulting from a misunderstanding of the relationship of symbols or groups of symbols to their meaning in a given language;
- Syntactic error – a violation of the structural or grammatical rules defined for a language.

However, errors are not only what is mistakenly introduced in the programming phase. One should make a distinction between software error and design error. The error above defined is a software error, which leads to software fault and software failure. On the other side, there are design errors, also called project errors, which are produced during the phase of specification, preliminary design or the final design [5][6][7].

2.1.2 Fault

According to IEEE Standard Glossary, the fault can be defined in two ways, regarding the software and hardware. In the case of hardware devices or components, a fault is a defect in that specific device or a component, for example, a broken wire, or a short circuit. On the contrary, fault in the software is defined as an incorrect step, process or data definition in a computer program.

The fault is the manifestation of the error in the program, it appears when an error is introduced in the software. In software engineering, the fault is often referred to as a bug. In fact, a bug is an error in the coding, which causes the program to behave in an unintended manner. It can prevent the software to perform given action or to execute one or more instructions in the wrong way.

Faults or bugs are usually found in the testing phase, by the software developers. In most cases, the fault can be identified and removed. Nevertheless, it is not uncommon for a fault to be discovered when a failure occurs. Indeed, the cause of the failure is fault [5][6][7].

2.1.3. Failure

Failure is the inability of the system or component to perform its required functions within specified performance requirements. It means the software or the system doesn't satisfy the client's request. Failure can also mean unexpected software behaviour recognized by the user. There can be defined several levels of severity of the failures, which can vary from a system to a system. The levels are the following: catastrophic, major or minor, depending on how do they influence the software.

One says that the system is in failure mode, when it operates slowly, gives incorrect results or terminates the execution of the program.

There is a linkage between the abovementioned terminologies, which is more effectively seen in Figure 3. An error in the software provokes a fault, while the fault provokes a failure. And the opposite is also valid: the cause of the failure is a fault, and the cause of the fault is the error [5][6][7].

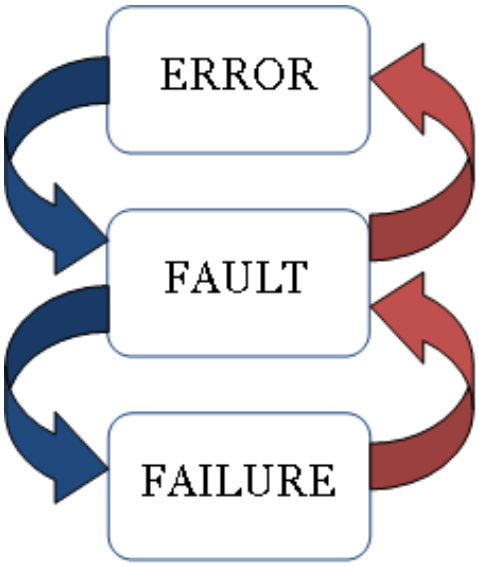

Figure 3: The connection between error, fault and failure

2.2. Software error→ sporadic and intermittent faults

As it has been already explained, software error defines the misunderstanding or misconception of the software developer, where the developer can be a software engineer, programmer, analyser or tester. A software error produces system behaviour that doesn't satisfy the requirements which were previously established. The reasons for these errors can be many among which: wrongly defined or misunderstood requirements, defects in the logic or they are of semantic and syntactic nature. On the other side, the outcome of a software error is a fault.

There are different types of faults, which can be divided according to different categories. Some of them are permanent, which means once occurred, the software will be unable to recover and will fail. Some of them are temporary, where the software is not working properly for some time, often in seconds or minutes, or the faults even are only instantaneous, but after that, it resumes as nothing happened. Some faults occur commonly and can be easily predicted, and/or removed. Moreover, there are faults called sporadic which are extraordinary, uncommon, sometimes unique, or at least happen rarely than the common ones. The sporadic faults occur once, repeated at regular rates or appear at non-regular intervals.

2.2.1 Intermittent faults

Intermittent faults are sporadic faults that cannot repeat easily because of their complicated behavioural patterns. Since they appear and disappear unpredictably, it becomes challenging to understand the origin of their causes. Intermittent faults are important because their repair increases the maintenance cost of the system [8].

Intermittent faults are considered under the NFF (no fault found) category. It means a failure occurred, but the fault that caused this failure cannot be detected, nor removed. Or, if tested, under the same condition, the system won't fail, and the fault won't appear. Intermittent faults and the NFF

problem are very important for companies, industries, especially transportation, avionics, nuclear plants; because they can lead to catastrophic failures. Those failures are expensive for companies and can even destruct their reputation. This type of fault can appear in both hardware and software, but in this thesis, only the software intermittent faults will be considered.

Intermittent faults appear in a remarkably huge number of software-based systems, but the end-user may never encounter them. The software intermittent failures are usually caused by a memory leak in the processor, change in the processor load, errors in the disk, or some unhandled exceptions. For instance, a low processor speed increases the probability of intermittent faults to appear, high processor load can contribute to failure in the system. On the contrary, with high processor speed and high memory capacity, these faults can happen unobserved. When an error is discovered during testing or debugging, and that error is not handled, can provoke an intermittent fault due to unhandled exception. An example of intermittent software fault is the freezing of the computer when it is immobilised to perform the requested task. After that, the software on the computer either crashes or continues to work as nothing happened.

The testing process of intermittent faults takes into consideration five techniques: deterministic replay debugging, fuzzing, HVTA (high volume test automation), load testing and disturbance testing. In the first step, the roots of the failures are determined, then in the fuzzing test, invalid and random data are inserted, and the reaction of the system is observed. While the fuzzing is good to detect data corruption, memory leaks and crashes, HVTA testing is good for the detection of faults related to buffer, stack overflow and timing failures. During the load test, the limit of the software is tested with stress and volume tests. In the last technique of testing, the software is studied in the presence of physical faults.

Another challenge regarding the intermittent software faults is that they depend on the hardware components or can appear due to hardware faults. That is the reason why in some cases these will not appear when the same

software is tested in a laboratory. Therefore, the challenge for software engineers and developers is to provide a solution to mitigate the faults and to make the software more reliable and less prone to faults [9].

Chapter 3 - Life cycle of the hybrid system

In the industry, as well as in engineering, there is the concept of the lifecycle of the product. Regardless if it is software or hardware product, the lifecycle starts when the product is initialized and ends when that product becomes inoperative. The System Development Life Cycle (SDLC) is a multistep process of planning, creating, testing, validation and evolution of the system until its disposal. The abovementioned process can be applied to a variety of hardware systems, many software ones, or hybrid systems which are a combination of the previous two [10]. Particularly important are the last ones, since nowadays every electronic device has integrated software, and the software and hardware are constantly interacting between themselves. Examples of hybrid systems are also the electronic control units (ECUs) in the IVECO vehicles, where the interaction between the integrated software and the hardware determines the behaviour of the given ECU.

3.1. SDLC phases

There are numerous SDLC approaches, but generally, each of them includes a series of the following phases: [11][12][13]

1. Requirement analysis or planning;
2. Design;
3. Development or coding;
4. Implementation and testing;
5. Integration and deployment;
6. Maintenance.

In the initial phase, the client (organization, company etc) establishes the requirements for the system. Under the requirements are the goals, services,

purpose and constraints of the system. These are analysed, defined and planned in detail and consequently documented. The analysis should answer to the questions regarding the economy, law, time, or rather does the budget satisfy the requirements, does the product faces any legal issues, does the project have a due date, etc. Early planning can foresee issues and problems which can corrupt the system and resolving them in this phase contributes to the economic management of the company.

Secondly, both hardware and software are designed following the previously defined requirements. The detailed requirements evolve into detailed system design and the architecture of the system is settled. Moreover, this phase is divided into high-level design (HLD) and low-level design (LLD). In the first one, every component of the system is named and briefly described, its functionalities are given and the interaction between components is established. On the other hand, LLD includes the design of the functional logic of the module, databases with type and size, list of error messages and complete input and output for each component [12].

In the development or coding phase, as the name suggests, the code is written, compiled, refined. All that was designed in the preceding phase, in this one is realized, such as inputs, outputs, databases, libraries etc, with a chosen programming language. It requires time to finish this phase, since it is sensitive, and many errors can be introduced by the developers.

The next phase is the implementation of the written code in the units of the system and then to test them. Taking into consideration the hybrid system, not only each component of the hardware is referred to as a unit, but also each program unit of the software. Unit testing aims to check the functionality of each unit and to verify that it behaves as expected. During this stage bugs and faults can be discovered in the system, after they are fixed a new test (re-test) is performed.

"It is easier and less expensive to fix design errors early in the process when they happen [14]."

In the following stage of the lifecycle, integration and deployment, the units, both hardware and software, are integrated and deployed. Therefore, by accomplishing integration and system tests, the system is checked for errors, bugs or faults. At the end of this phase, the completed hybrid system is released and transmitted to the client.

The last phase in each of the SDLC approaches is the maintenance, and sometimes denominated as operation and maintenance. Usually, it is the longest phase where, after the system is being installed, it is observed until the end of its life. The activities that are performed during maintenance are: fixing the bugs which were not discovered in the preceding phases, upgrading the system to improve the services, changing the initial software by adding new services, replacing hardware or software components or add new ones. However, the latest modifications should also be tested in order not to crash the entire system. As the last activity of the maintenance phase, the system should be evaluated to check if it operates in the required mode.

3.2. SDLC model

In some of the lifecycle process models, it is possible two or more phases to be merged into one single phase, or a phase to be divided into two parts. Depending on how the phases are arranged, some of the SDLC models are:

- Waterfall model;
- Incremental lifecycle;
- Evolutionary lifecycle;
- V-diagram [15].

3.2.1. Waterfall model

The Waterfall lifecycle model is the first SDLC model to represent the development process of the system, proposed in 1970 by Royce [15]. It is called

a waterfall since the blocks are arranged as a cascade, as shown in Figure 4 [15]. It is a very rigid model and the succeeding phase cannot start before the preceding is concluded. At the end of each phase, it is reviewed, and the developer decides if the system is eligible to proceed with the further phases of its lifecycle. As it can be seen the system is not flexible, and any change in the requirements by the side of the client, after the first phase is completed, can corrupt the entire system. Consequently, this model is applicable only when the requirements are well comprehended [15][16].

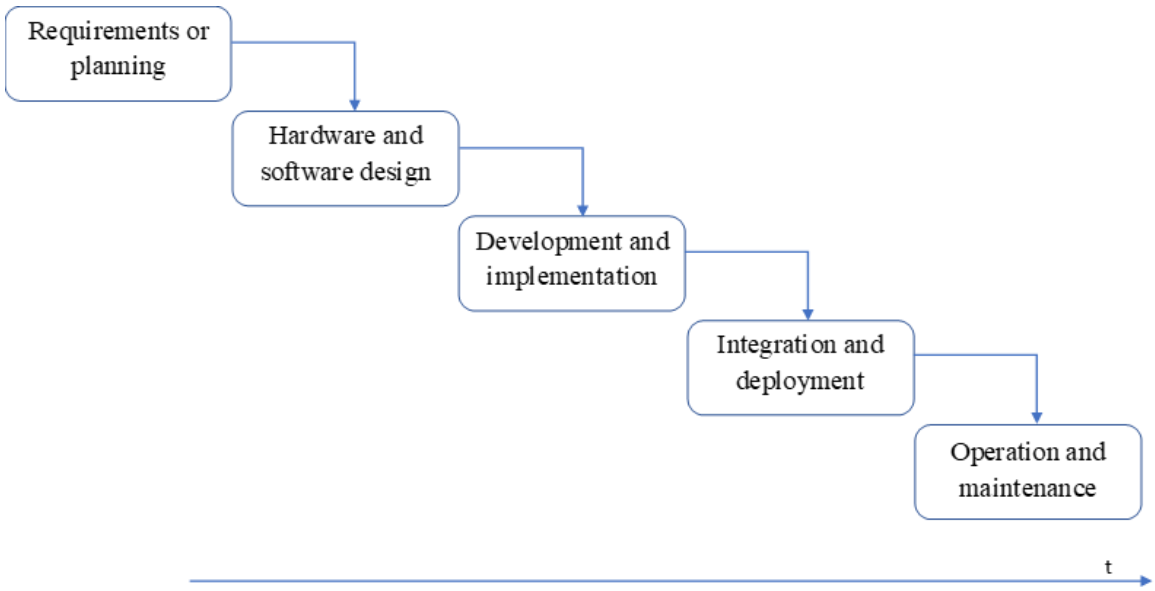

Figure 4: Waterfall lifecycle model

3.2.2. Incremental model

Since the Waterfall model has been criticized due to its inflexibility, other models have been proposed, among which also the Incremental lifecycle model, shown in Figure 5 [15]. This model was suggested in 1980 by Mills [15]. If the development phase requires a huge amount of time, it may be divided into smaller pieces, called increments, and only one increment is developed and implemented in time. Usually, the most important services are developed first, and the customer can have a clearer idea of what to expect of his system, how it behaves, and how can further upgrade it. The advantages of the Incremental approach are many, such as: it allows the clients to postpone their final

decisions and expectations of the system they are requiring; the client can exploit the system before it is fully completed and thus, can already have a benefit of it; decreases the risk of failures of the system and can anticipate the errors and faults which can lead to a failure of the system [15][16].

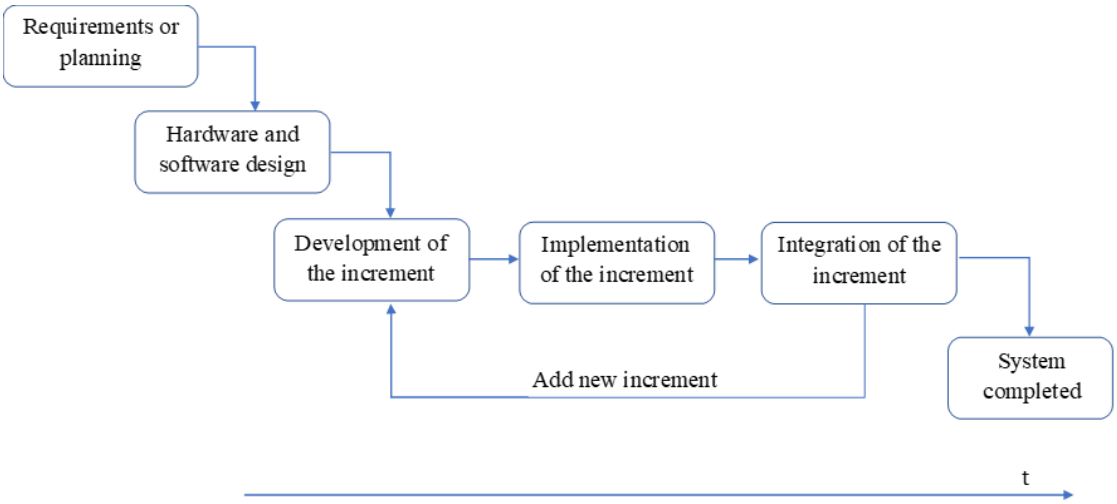

Figure 5: Incremental lifecycle model

3.2.3. Evolutionary model

Another SDLC model is the Evolutionary lifecycle, shown in Figure 6 [16], where all the phases can be repeated. With this model, a variety of different versions of the given product are developed, until the suitable one is made. Each of these versions is a prototype, and by studying the prototype, both the customer and the developer can review the operability of the system, and then further modify, develop or upgrade it. There are two types of prototypes: throwaway and evolutionary [16] (also called exploratory development). The throwaway prototype is fast produced and faster discarded. Its scope is to understand better the client's requirements, to test a specific part of the system, either hardware component or software program, and to implement the knowledge into the next prototype. On the other side, the evolutionary prototype is not discarded but grows to the completed product. Firstly, the components which are understood are developed, and then the following are

added one by one until it arrives at the final system. Differently to the throwaway prototype, the development of an evolutionary prototype is not that fast [15][16].

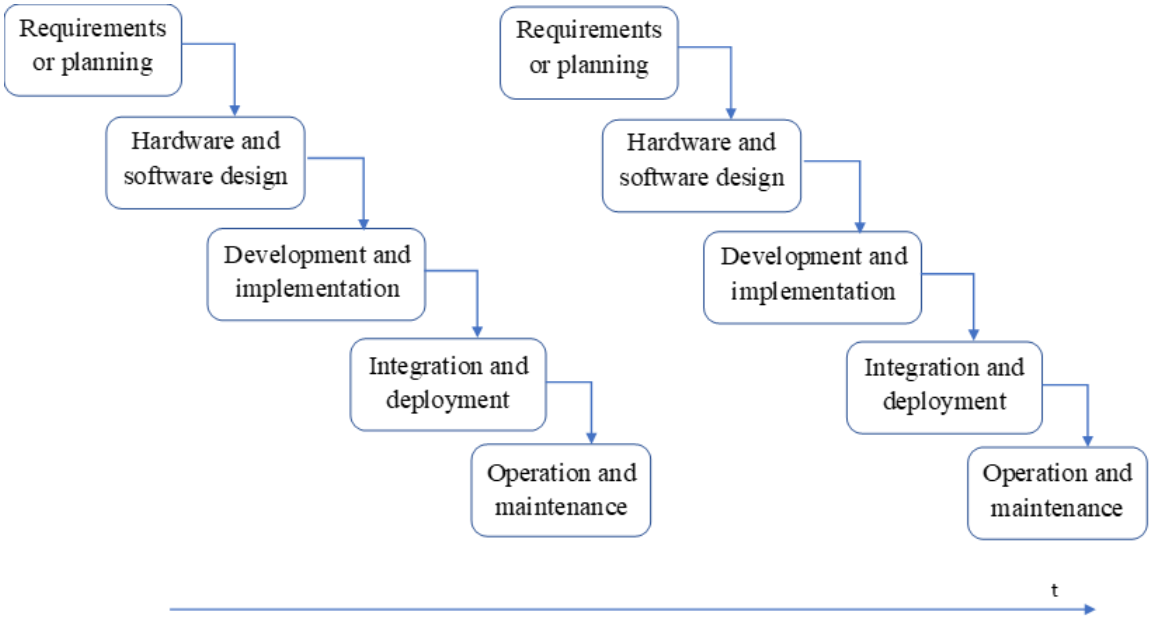

Figure 6: Evolutionary lifecycle model

The Evolutionary lifecycle model has advantages and disadvantages. It gives an early view of the functionality of the product; requirements can be modified at any time. However, it may be cost-ineffective, since many prototypes may be required, or time-consuming.

3.2.4. V-model

The V-diagram model, or shortly V-model, got its name because the phases are progressing linearly until the coding phase, then are tilted and continue upwards, forming the letter V, as shown in Figure 7 [15]. On the left side is the verification process, where the system is designed according to the requirements and regulations, while on the right side is the validation process,

where the system is implemented and tested to check if it satisfies the requirements. The design phase, in this approach, is divided into system design, detailed hardware and software design, and detailed unit (module) design. On the right side, after coding, is the unit implementation, then hardware/software integration, and the last is system integration. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. As it can be seen, there are three elements on the left, three on the right, and accordingly, there are three tests that are connecting the elements on both sides:

- unit testing: each unit is tested separately of the other components of the system;
- integration testing: the test is applied on units combined in the way to form a subsystem, isolated from other subsystems;
- system testing: the test is performed on the entire system to check if it meets the predefined requirements.

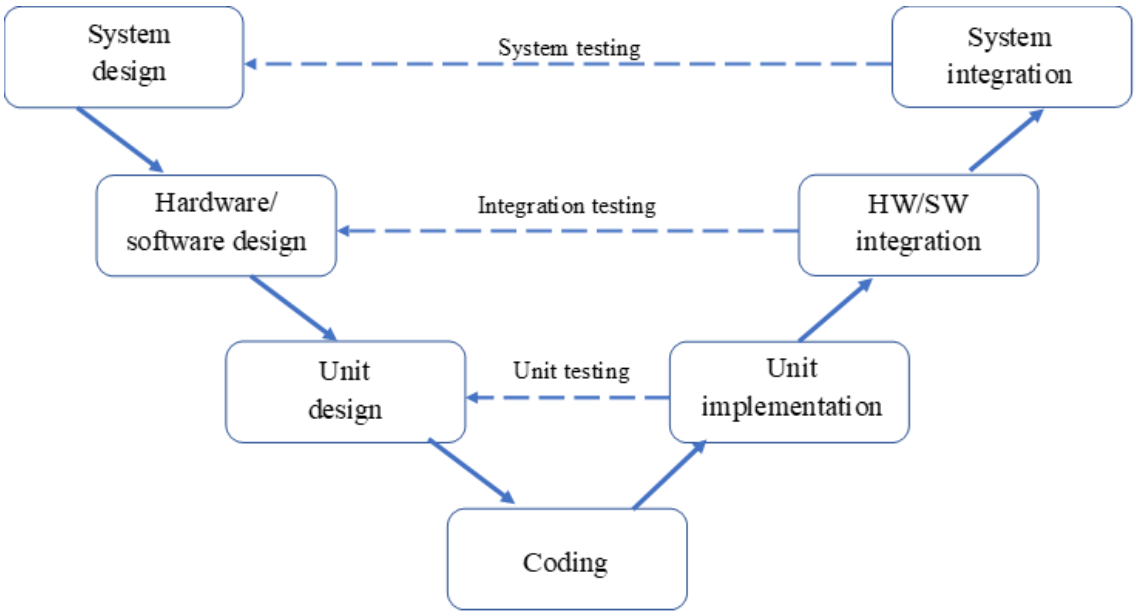

Figure 7: The V-diagram lifecycle model

As the aforementioned lifecycle models, also this approach has its pros and cons. Since the tests go alongside with the design, it decreases the risk of errors and faults. With the V-diagram the system can be better understood, so the desired quality can be provided, and this model is important for different system testing, which will be explained in the following chapter. However, this model has been criticized as inflexible, just an update of the waterfall model. And in the V-model the operation and maintenance phases are not included, so they must be performed separately [15][17].

Chapter 4 – Verification and validation of the software

The terms verification and validation have already been considered in the V lifecycle model in Chapter 3, where the verification processes where those on the left side, such as: system design, hardware and software design and unit design, while on the right side are the validation ones such as: unit implementation, hardware/software integration and system integration.

In fact, verification and validation, often found abbreviated as V&V, is the set of processes to check and analyse software if it satisfies the requirements specified by the client. The words sound similar and can be easily confused, but they represent a different thing. One way on how to distinguish them is expressed by Boehm in 1979, where the verification answers the question: “Is the product built in the right way?”, and on the other side, the validation gives an answer whether the right product is being built [15]. Other differences are due to the mechanism used to implement these processes, or in other words, the verification is a set of static processes, where the design and the program are checked without executing the code, while the validation of the product is a dynamic practice, and to test it the code must be executed. Having different mechanisms means having different methods to verify and validate the software, namely the methods in the verification processes are inspection, automatic analysis, walkthrough etc, despite the methods used in the validation which are black box testing, white box testing and other testing techniques. Undoubtedly, the verification is firstly done, and it is followed by the validation [18].

4.1. Planning of V&V

Being an expensive and very important process, the verification and validation should be planned carefully, in order to achieve a great deal of the methods to verify and validate the software, and thus to handle the cost of implementing this process. Respecting the same reasons, the earlier the planning starts, the better.

The role of the planning is to define which standards, methods, procedures to be used in the phase of inspection, which ones in the phase of testing and to establish a well-defined test plan for the final software product. The commitment to the planning should be proportional to the importance and criticality of the requested product. The scope of the test plan is not only to describe the testing to be used rather to be a guide for the software engineers to follow the previously established procedures. The software test plan is not a static document, but dynamic. It is also an incremental document, because the plan may be subjected to modification in case they are needed.

The structure of the test plan for the software product includes the following elements:

- Testing process: where the description of the process is provided;
- Requirement traceability: test plan should not conflict with the already settled requirements;
- Tested items: registering the items which are going to be tested;
- Testing schedule: the testing is scheduled as a part of the overall development schedule;
- Test recording procedures: where the results of the finished tests are recorded;
- Hardware and software requirements: hardware and software items needed to perform the tests are listed;
- Constraints: constraints that may corrupt the testing have to be anticipated [15].

4.2. Verification

Verification is a static mechanism with the aim to provide high-quality software. The activities to perform this task include verification of the documents, of the code and verification of the design. In order to complete the verification, the software engineer uses several methods such as: inspection, automatic static analysis and the “cleanroom” as a formal method of verification [18].

4.2.1. Inspection of the software

Software inspection is the method of verification when the code and the design are reviewed by so-called inspectors, which are people different than the ones who developed the program. The inspection gives a new perspective; a second pair of eyes is going through the program and it can detect an error easier than the original developer. This is since the developer is not suspecting the correctness of the program. Also, in this way, ideas, knowledge can be interchanged by the developers and inspectors and most importantly, errors can be detected early in the process.

There are two types of inspection: formal review and walkthrough. The first one is restrained by well-defined rules, which means that: the roles are well-defined; all participants need to be prepared for the meeting; the meeting is formal one where the program or the process is discussed; and the follow-up processes are formal, too. On the other hand, for the walkthrough: the participants don’t need to be specifically prepared; no formal follow-up is needed; and, the author is the main participant in the discussion. There is a trade-off between the cost and the efficiency regarding the two before-mentioned inspection techniques: walkthroughs cost less, but with the formal inspection more errors are found in a program.

The main roles in the inspection process are author, moderator, recorder and reader. The one who organizes the review is the moderator. His role includes activities such as, to keep an eye on the discussion, to control if there

are follow-ups. Being a moderator means managing the review, so the person has to have knowledge for the selected topic, to be a good facilitator, respected etc. His role is different than the one of the recorders, which includes capturing a log of the inspection process.

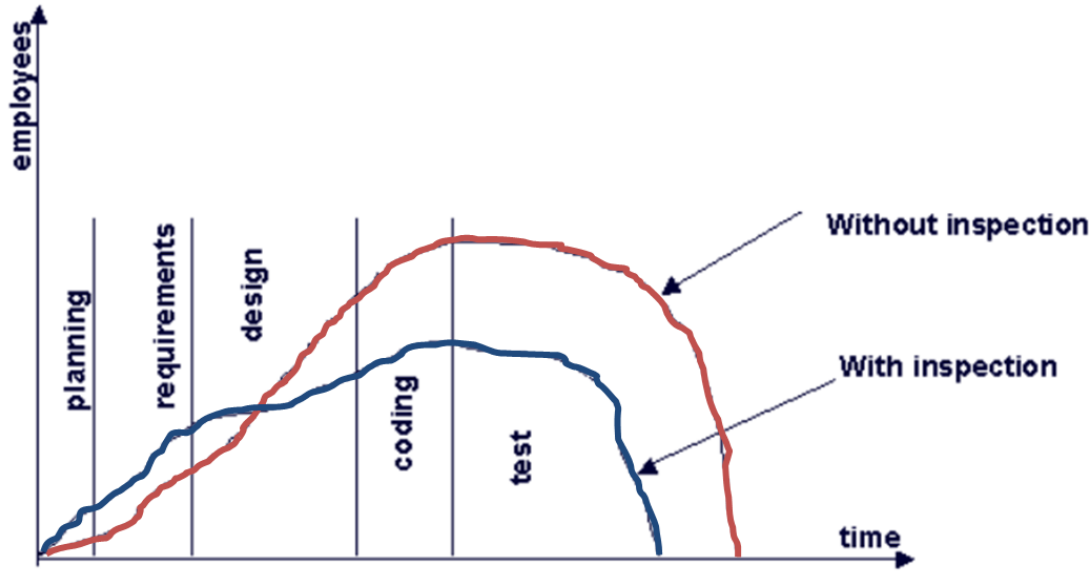

Figure 8: Inspection of the software – Fagan's diagram

The main activity that the author has to complete is to describe the logic behind the completed work. The same person cannot execute the other roles since the author cannot be as objective as a person that sees the code for the first time. However it is important the participation of the author in the meetings since he can consider the different perspectives, he can answer the questions of the other participants and can share his knowledge with the others.

The reader is the one who presents the material and comments for each section. He presents his point of view or his interpretation of the code that the author wrote. In case these interpretation differ from other participants' interpretations, including the author's, is a good argument to start a discussion and to overcome the differences.

Figure 8 [19] represents Fagan's diagram, the so-called snail curve. It shows that by applying the inspection method, in beginning there is a need for more people resources (employees). But, over time, in coding and test phase fewer people resources are needed and the testing is completed faster than without inspection [15][20].

4.2.2. Automatic static analysis

The automatic static analysis is done with special software, so-called automated static analyzers. With the aim of these software tools, possible faults can be detected by an automatic scan of the code of the program. These tools can check the statements if they are well-formed and can compute possible values for the given data. This type of verification is way cheaper than the inspection since no specific people are required for each task, way faster since everything is automated and in that way is better than the inspection. That is one of the reasons why it is easier to perform the automated static analysis as a verification process in the development stages than the formal reviews. Nevertheless, the automatic static analysis is not as effective and detailed as inspections are and can lead to many false positives. That means, the analyzer can highlight potential errors, which in reality are not and it is time-losing. The purpose of this verification process is to point out the most usual errors, which are classified into five fault classes: data, control, I/O, interface and storage management faults. All the possible fault checks are given in the following table [15].

Table 1: Automatic static analysis - Fault class & Static analysis check

FAULT CLASS	STATIC ANALYSIS CHECK
Data faults	Variables used before initializations
	Declared variables, but unused
	Twice assigned variable, but unused between assignments
	Array bound violations
	Undeclared variables
Control faults	Unreachable code
	Unconditional branches into loops
Input/output faults	Variables output twice without assignment
Interface faults	Mismatch of the parameter type
	Mismatch of parameter number
	Unused result of the function
	Uncalled functions and procedures
Storage management faults	Unassigned pointers
	Pointer arithmetic
	Memory leaks

There are three stages of performing the static analysis checks, which are characteristic error checking, user-defined error checking and assertion checking. With the first checking, the most common errors are highlighted, which represent 90% of the errors[1]. This makes the characteristic error checking very simple, cheap and effective. The user-defined error checking, as the name suggests is the approach designed by the users of static analysis for exposing the errors. They usually use error patterns, like following the priority, first method one must be executed, then method two and not in the opposite way. The third

[1] Zheng and his collaborators (Zheng et al. 2006) analyzed a large code base in C and C++. They discovered that 90% of the errors in the programs resulted from 10 types of characteristic error [15].

stage is the most capable one, where formal statements (assertions) are part of the code of the program. An example of these assertions is the given interval for the value of a variable. With the aim of the assertion checking, the analyzer clarifies where the value is inoperative.

4.2.3. Cleanroom software development

The "cleanroom" method is a formal verification method developed in the 1990s (Mills, Dyer et al., 1987; Cobb and Mills, 1990; Linger, 1994; Prowell, Trammell et al., 1999 [15]). The approach in this method is to avoid the defect rather than to remove it. The cleanroom method is established on five procedures to reach the goal of zero-defect software:

- Formal specification;
- Incremental development;
- Structured programming;
- Static verification;
- Statistical testing of the system.

Firstly, the software, as the name suggests, is formally specified by using the state-transition model. The state-transition model is a type of model where are highlighted the responses of the system to a stimulus. Then, each increment of the software is developed individually, and after that, it is also individually validated. The following strategy is structured programming, where the process of stepwise refinement is used. It means that the software is developed in different steps, or levels of abstraction and the software is refined incrementally with each level, or in other words that each increment is more detailed than the previous one. After that, the software is submitted to precise and pretty rigid inspections of the software. And the last strategy is statistical testing, where the reliability of the tested software is established. The way that the aforementioned strategies of the cleanroom process are coordinated between themselves, is shown in the model designed in 1994 by Linger, shown in Figure 9 [15].

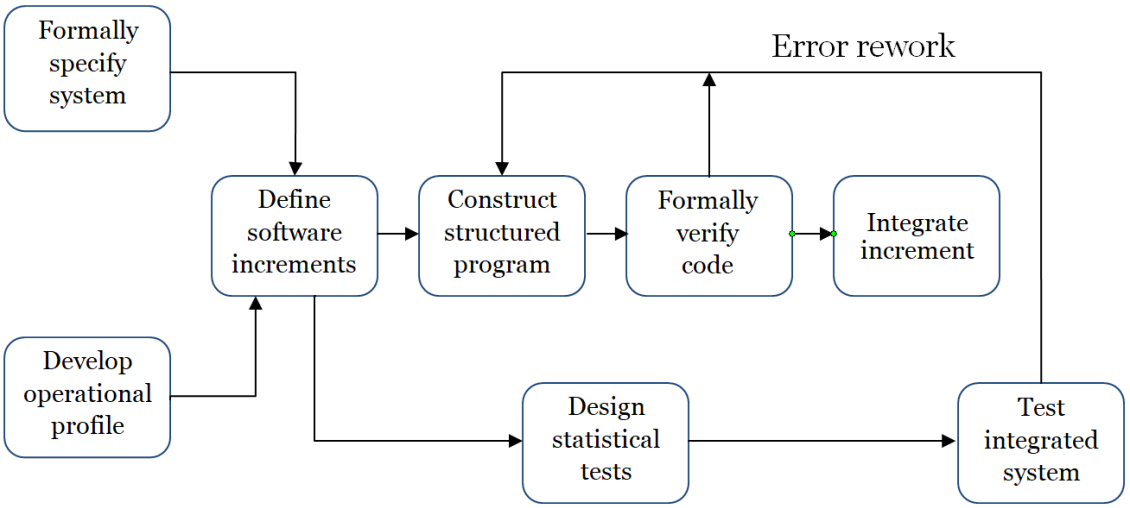

Figure 9: The "cleanroom" process

4.3. Validation

Validation is the dynamic part of the V&V process, where, by executing the code the program is tested. It highlights the errors that cannot be discovered by the inspection and the other verification methods. The methods in the validation part of V&V are involving different types of testing and it is on a higher level, which means also the testing team is included to perform the tasks.

In different stadiums of the development of the software, there exist different types of tests, such as validation test and defect test. The first test aims to prove that the software satisfies the requirements and operates accurately. The defect test, on the other hand, discovers errors and faults in the code of the program and the cases where they manifest. It is executed in the way that the test highlights the incorrect way of operation [15][18].

4.3.1. Testing

According to the IEEE Standard Glossary of Software Engineering Terminology, testing has a double definition, as follows:

- IEEE Standard 610.12-1990: Testing is defined as a process where the software or its components are working under specific conditions and this operation is observed, documented and evaluated [5].
- IEEE Standard 829-1983: Testing covers the analysis of software to detect discrepancies between existing and required conditions, such as errors, bugs, etc, and to evaluate the characteristics of the software [21].

Testing is not a single activity, but a process of different activities to achieve the desired goal, software that is reliable for its customers. The activities include planning of the testing, controlling the process, analysing and design of the testing, then implementation, execution, evaluation and report, and to conclude with test closure activities. The planning has to be done thoroughly, to explain the objective of the testing, to determine the resources needed to complete the testing etc, while control activity checks if everything is executed according to the plan and in the given deadline. Analysis means to analyse, to check the basis of the tests, the requirements and other conditions that need to pursue the design of the test and the environment where the test is set up. Execution, as the name suggests, means to execute the code in an appropriate program. The implementation, though, means to create test cases with specific data for each, to re-execute the failed test after they are being fixed, to confront the expected results with the obtained one, etc. Both evaluating and reporting activities are before closing the test, where the software and the code of the software are taking place evaluated according to the evaluation criteria defined in the test planning, and everything is documented and reported. Test closure is performed when the software is positively evaluated by the testers and ready for distribution to its customers. Moreover, the test can be finished due to the cancellation of the project, achieving a specific target and so on. The main task is to give an evaluation mark to the testing, find its advantages and disadvantages, and learn lessons for projects to be performed in the future.

There are different types of testing, and they can be divided into different groups depending on the criteria used. Moreover, the same type of testing in two distinct pieces of literature can be defined under different categories. Then, some of the types of testing are defined as testing methods, testing strategies, testing techniques, testing levels, subcategories of other types of testing, etc. In this chapter, some of these types of testing will be explained, while some of them will be only mentioned [15][22].

4.3.2. Testing methods

The most used testing methods to perform for software testing are black box and white box testing. There is another one, as a combination of the previous two, called gray box testing (Figure 10 [23]).

Black-box testing

Another name for this type of testing method is functional testing. The testing is performed without knowing the internal code of the tested software. Even though it is executed by the testing team, it is using customer perception. It means the testers are only interested in the outcome according to the input values and not what is done inside, which programs, lines of code are executed or how the software processed the result. The name black box derives from a metaphor where the software is seen as a black box, and the tester cannot see what is inside. In reality, there are many examples to show how this method works, such as Google search engine, ATM machines and others. People are not interested in how Google is combining the sources or what is going on inside an ATM machine, but to find the result of their research and to get the exact amount of money they inserted for withdrawal.

Black box testing uses many techniques to perform the testing, such as equivalence class; domain tests; boundary value analysis; orthogonal arrays; decision tables; exploratory testing; state models; and all-pairs testing.

Like every other type of testing, even this method has its positive and negatives sides. Considering its way of operating, it is pretty efficient for larger parts of code, and non-technical testers can perform the testing. However, only a certain number of test cases are tested, thus it is not covering the whole software. Even if there is a possibility to improve the coverage with a minimum number of tests, having non-technical testers and working blindly, without knowing the code, can be a difficult task.

White-box testing

In contrast to black box testing, in this method, the internal code is relieved to the tester. The tester selects inputs and considers the possible outputs for each input going through the lines of code. The idea of white box testing is to improve the software, its design, efficiency and usability, to make it more secure. Another contrast to black box testing is that in this method, non-technical testers with limited knowledge are not accepted. Therefore, the testers have to know programming languages and their implementation in the software, or software components, which they are testing. If in the previous method, the testers were compared to people withdrawing money from an ATM, here they are considered as surgeons or car mechanics, since they have to have knowledge about the organs in a human body or different parts of the car and how they are connected.

To perform the white box testing, the tester firstly has to understand the code and then to create test cases and executes them. The idea is to cover as much as possible, reaching 100%. This is done by using the white box testing techniques, such as statement, branch and path coverage. In other words, it means the tester has to perform a minimum amount of tests to cover all the statements, all the branches at least once. With the testing of all the possible paths, it is guaranteed that every branch and every statement is covered. Types of testing that are using the white box method are unit testing and memory leaks testing, of which the first one will be explained later in this chapter.

The white box method, known also as glass box, clear box or open box testing has its advantages and disadvantages. The advantages are the coverage and thorough testing. Knowing the code is helpful for the testers to minimize the number of test cases and can lead to optimization of the code. Nevertheless, these thorough tests are cost-inefficient and time-consuming. Depending on the size of the software, 100% coverage may be unreachable, and with the performed tests an error is missed.

Gray-box testing

Since the combination of the black and white colour gives gray colour, even the combination of black box testing and white box testing leads to gray box testing. In this testing method, the tester has limited knowledge of the code, which means only some lines are revealed. The test cases are developed according to the provided structures and algorithms of internal data, but testing is treated as black box one. The pro of the gray box method is the combination of benefits of black and white box testing, it is non-intrusive, and testing is unbiased. Non-intrusive testing means based on functional characteristics and not on the source code; and unbiased means the tester and the developer can be two distinct persons, so the tester does not know the internal code. But, limited access leads to limited test cases and limited coverage [22].

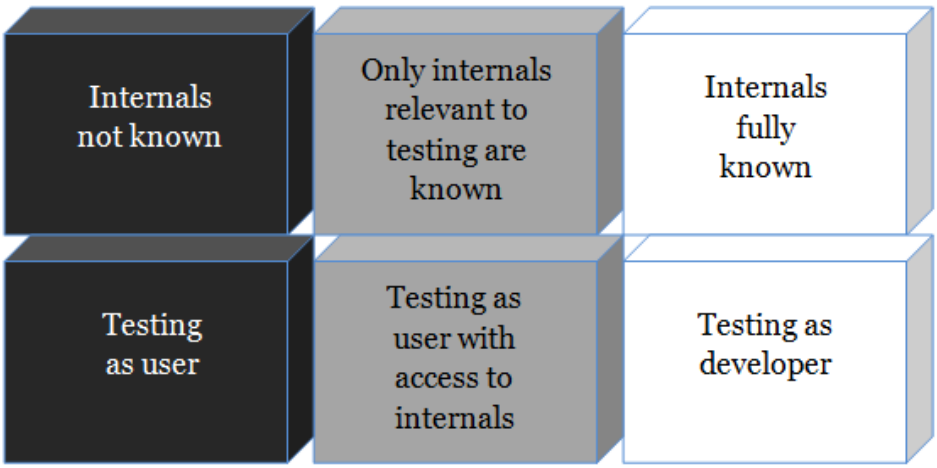

Figure 10: Conceptual difference between the three testing methods

4.3.3. Producer/user levels of testing

According to the timing of the testing, the types of testing are divided as testing on producer level and testing on the user level. The types of testing on producer level are unit testing, integration testing and system testing, while on the user level it is user acceptance testing, with its subcategories alpha and beta testing.

Producer level – Unit testing

Unit is the smallest component of the software, and therefore the smallest part to be tested. Unit testing aims to test each separated unit to check its correctness by using a white box method of testing. These unit tests are executed by the software developer and have to be seen as routine calls with different inputs. Since even the simplest software is a system made of many units, the number of tests is proportional to the size of the software. Due to this, it is advised to perform automated unit testing, where the software developer automates his tests by using a test automation framework. The framework aims to accelerate the testing up to a few seconds and they are executed after every change is being made. It is composed of three parts that are: setup, call and assertion. Firstly, the testing is starting with the specific test case, the given inputs and the expected outputs, then the unit is called to be tested and in the comparison of the final part of the expected and obtained result is done. The test is failed in the case there is a discrepancy between the expected and obtained results. Even though the testing can have a positive outcome, the unit may depend on other units and to give different results where these two or more are integrated. Therefore, another testing of higher level is needed, such as the integration testing.

Unit testing is essential for the error catching since the code is tested every time a change is made. Thus, it reduces the cost of bug fixing and the debugging process is simplified. When the testing fails, only the last change is prone to bugs, so that one is debugged [15][22].

Producer level – Integration testing

Integration testing is performed when several units are assembled and are operating as a group or subsystem. It is carried out after each unit is being tested individually and with a positive outcome. Integration testing is one of the most important testing since the interaction between successfully tested units has to be executed. It means to find an assembling language for communication and to discover the faults that can come up due to the interaction between those units.

The strategies to perform integration testing are big bang integration testing and incremental integration testing. Big bang testing is performed when all subsystems are integrated and they are all checked at once. This type of testing is only favourable for the waterfall software development lifecycle model. The incremental integration testing is performed in the way that units are integrated one by one. It starts with a couple of units and after successful testing, an additional one is added. That is why it is called incremental since each added unit is the new increment. It is easier to find the bugs with incremental than with the big bang approach since the bugged unit will be the last increment [22].

Depending on the way in which the testing is conducted, the incremental integration testing is further divided on:

- Top-down testing;
- Bottom-up testing;
- Sandwich integration testing.

The top-down technique as the name suggests, starts from the top, from the highest point and gradually continues to the lower ones. It means that the highest level unit is the first one to be tested, and then the lower ones are added one by one, as increments, until the desired subsystem is constructed. Actually, in the beginning, when the lower modules are not yet available for the integration, the testers are considering the usage of stubs. Stubs are just pieces of code, a dummy program to which the testers are giving input and are interested in the obtained response. Even though the explanation of stubs is

simple, in the practice, they are a very complex system, and their complexity is proportional to the complexity of the system or subsystem. It is shown in Figure 11 [15]

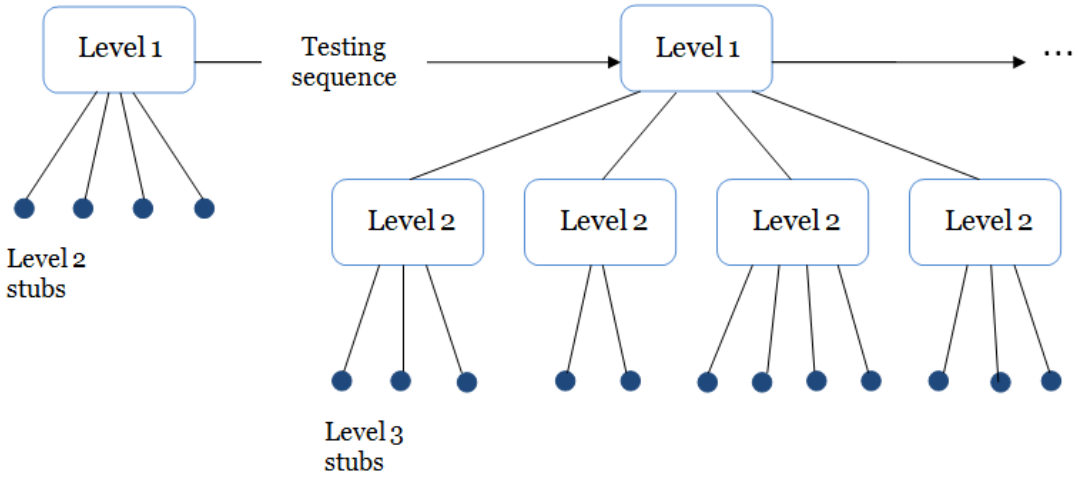

Figure 11: Top-down approach

The bottom-up approach (Figure 12 [15]), differently from the top-down approach starts from the lowest levels and gradually goes to the upper levels. In both approaches, the moving up or down continues until the whole subsystem is tested. In the case of bottom-up testing, when the upper levels are not yet developed and integrated, a simulator is needed to call the lower levels. Similarly to the stubs in the top-down approach, there are drivers in the bottom-up approach. Another similarity is that the drivers are dummy programs as stubs with the sole purpose to call lower level when there is no calling function.

The advantage of this approach is when there is a fault in the lower levels, their detection is fast and can be easily fixed. But the subsystem is inexistent until the highest level is integrated together with the lower ones. The disadvantage is if a fault occurs at the highest level, it will not be detected until the end [15].

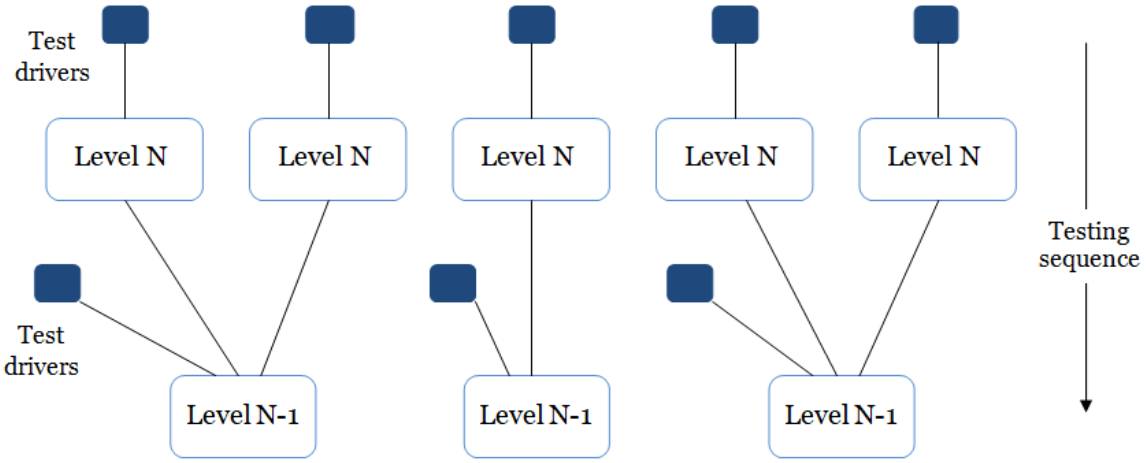

Figure 12: Bottom-up approach

The sandwich integration testing approach is a hybrid approach that combines the advantages of the previous two approaches. It uses both stubs (top-down) and drivers (bottom-up). There are three layers: the main target layer; a layer above the target layer; and a layer below the target layer. The testing is mainly concentrated on the main target layer, but before reaching it, the stubs are needed to test the user interface in isolation and the drivers are used for testing the lowest level functions.

The advantage of the sandwich approach is that it can perform parallel testing, saves testing time and therefore, it is highly recommended for large and complex systems with large plentiful subsystems. However, these advantages have a price, making the sandwich approach a costly one [24].

Producer level – System testing

System testing is performed on complete, integrated systems where their units and subsystems are working in harmony. System testing provides information if the previously integrated units are compatible and correctness of their interaction and data transfer. It is similar to integration testing, but it is

important to be performed since the subsystems that are working correctly alone may be dependent on other subsystems and are producing unexpected output. Also for larger systems, there is a possibility that different developers developed the subsystems, thus they have to be integrated among themselves in order to communicate in the correct and expected way. All systems have emergent behaviour (side effects), that can be planned or not. Some of the functionalities of the system are being observed when the components are assembled. The idea of the system testing is to control and confirm that the system is operating accurately, or in other words, that system does what it is expected of it. The testing with different test cases is useful here since the communication between two integrated components and the interaction between them are being observed. The system testing helps to overcome the misunderstanding and to minimize the side effects. The number of possible test cases is proportional to the size of the system, and the so-called exhaustive testing is impossible. Exhaustive testing is when every possible line of code is executed, but for a complex system, it is ideal to select a subset of test cases. The criteria for the subset are different from one software company to another. For example, both correct and incorrect input values can be used to test the functions of the system, testing a combination of functions and so on. Unfortunately, in contrast to unit testing, automated testing for the system is complex to be executed [15].

The system testing can be divided due to its functionality, so there are functional techniques to cover the quality of the software and non-functional techniques for the performance of the software. With the functional types of testing, the software is tested against its functional requirements. Therefore, the techniques of functional system testing are: unit testing; integration testing; smoke testing; sanity testing; system testing; regression testing; and acceptance testing.

On the other hand, non-functional techniques of testing are executed to check the non-functional requirements of the software, such as security, stability, durability, usability, flexibility. The names of the types of testing are

according to each requirement, so there are usability testing; scalability testing; security testing; recovery testing; reliability testing; performance testing [22].

User level – User acceptance testing

Before jumping to the testing on the user level, there are other types of acceptance testing, such as contract acceptance testing and regulatory testing. With these types of testing, the functionality and performance are not controlled, instead, it is checked if the software respects the law in different countries or the articles in the contract.

Contrary to previous types of testing the user acceptance testing, abbreviated as UAT, and its subcategories are on the user level, which means the customer is needed to perform them after the system testing is successful, but before the product is being released to the market. User acceptance testing, known also as end-user testing, is a formal type of testing conducted having in consideration the requirements of the user to check if it satisfies the criteria of the customer. The outcome of this type of testing is either acceptance or discarding of the software product. The UAT is using the black box testing method, since the user is not interested in the internal code of the used software, but only in its performance. There are two types of UAT, alpha and beta. This testing is performed by both client and end-users, where the clients are the ones that own the developed software product. The need for this testing is due to the fact each entrepreneur has an idea for new software to be developed, but it is wrongly interpreted by the developer, or not well explained by the client.

The UAT follows after the unit, integration and system testing are completed and the bugs in each one are fixed. The UAT is done on the location of the client, following the testing plan. The testing has to be designed to check the criteria given by the client. After the UAT is designed, it is executed and the number of bugs is reported. In case there are bugs, they have to be fixed and then execution is repeated. This is a crucial phase of the acceptance of the system, since after the testing is done the destiny of the system is decided. The

UAT is finished if no critical defects remain unsolved, and it is business efficient.

User level – UAT – Alpha testing

Alpha testing is the first testing to be conducted, and it is usually happening in a laboratory by a team member, or other employees in the company where the software has been developed, or employees in the client company. After it is being completed, minor changes can be done to improve the performance of the software and then it is followed by the beta testing.

The positive sides of the alpha testing are that it simulates real-time behaviour, so the reliability and the robustness can be checked in the initial stadium. It also provides to its customer secure and high-quality software products. To pass with positive evaluation, the alpha testing has to confirm that the software meets the requirement of the client.

User level – UAT – Beta testing

Beta testing is the second phase of the user acceptance testing, where a sample of the software is released to real users in a real environment, not a laboratory. The users are giving their opinion and evaluation of the program, thus contribute to inputs in the design and the functionality. The evaluation of these testers is important to get the opinion for the software product before releasing it to the market and shipping it to the customers.

The advantage of this testing is the reduction of risk, failures and faults due to the evaluation of the beta testers. These evaluations and feedbacks are important for future update releases and are meant for the satisfaction of the customers [22].

Chapter 5 – New paradigms in the software verification and validation development process

The always increasing need to manage diversified activities of human life, through technologically advanced electronic devices, requires continuous improvements and greater guarantees of the software used in them.

Within the scope of software development, it has never been an issue to introduce new methodologies. Therefore, in the last 25 years, numerous different approaches to software development have been introduced, but many of them have been discarded. Only a few of them survived to be used nowadays. Software providers are being asked to combine two crucial features: to improve the quality of the software provided, but at the same time to reduce the overall cost [26].

The business management models had to adapt the recent "turbulent" conditions of production, characterized by immense competition and requirements for large production, velocity and quality. It is a context that brought changes to the nature of project management; accordingly, the principles of traditional management are not sufficient anymore.

Precisely, to help PMs (Product Managers) and company organisation in the management of "complex projects", advanced project management methodologies and techniques have been introduced. In the last years, a wide range of technologies was developed with the scope of making the software more reliable [25].

Even though there is no agreement for the exact meaning of the "Agile" concept, the same one provoked much interest among the professionals and lately, in the academic circles. Some proposals and ideas relevant for the modification, or revolution, to the management of software development lifecycle are illustrated in the present chapter [26].

5.1. Agile software development

The Agile method (or Lightweight methodology) originates in the IT around the '90s as contraposition and evolution of the Heavyweight methodology. It is an innovative method based on continuous interaction with the stakeholders, whose satisfaction is crucial for the success of the project and the development of the organization [25].

The "movement Agile" in the software industry was introduced in 2001 with the "Manifesto for Agile Software Development" published by professionals and consultants in the IT sector. As it has been stated in the Manifesto [29], the values that are respected are:

" *1. Individuals and interactions over processes and tools;*

2. Working software over comprehensive documentation;

3. Customer collaboration over contract negotiation;

4. Responding to change over following a plan. "

According to the Manifesto, the values on the left are valued more than those on the right side of the sentences, although both have to be appreciated.

The Agile method is not based on the idea of the classical linear design approach, but on the possibility to realize the project stage by stage, called "sprint". Each sprint is characterized by a new feature and is verified by the client in terms of satisfaction. The client meets with software developers and he is brought up to date with the work done. This is an iterative (and interactive) system that allows changes in the project to be performed easier, reduces production costs, and most importantly, it helps avoid unnecessary efforts and eventual project failure [25].

5.2. Extreme programming (XP)

This type of development methodology has arisen from the problems caused by the long development cycles in the traditional development models. It has started simply, as an opportunity to perform the work with practices that were found efficient in the software development processes in the previous years or decades.

Figure 5.1 shows a high-level view of the lifecycle of the XP development. Although the term phase is an immediate indication to the waterfall SDLC model, here the fact is that the phases can occur iteratively; which is evident in the figure, precisely, with the fact that it is possible to move back and forward between phases of planning, iteration to release and production.

The phases are not necessarily long; for example, the planning phase may only take a few hours. Besides, the XP teams do not typically think they are working in phases, but they think only of themselves working. Having said this, it is easier to think of the development as evolving one phase at a time, thus they are stated in the following way: exploration phase, planning phase, iterations to release phase, production phase, maintenance phase [28].

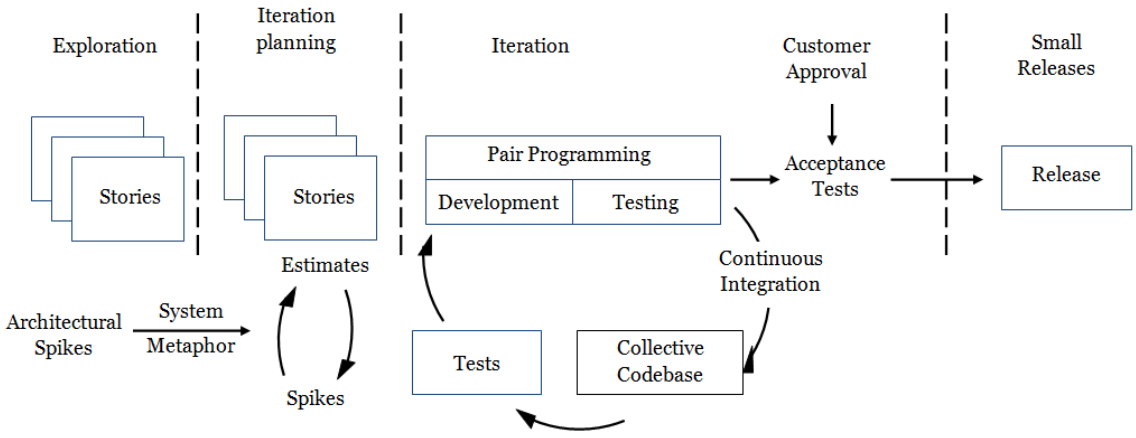

Figure 13: Extreme programming (XP) at a glance

5.3. Scrum

Scrum is the most popular Agile methodology, particularly indicated for complex and innovative projects. It is a framework, which means logic architecture, a support frame on which the software can be designed and realized. Specifically, it is a particular set of practices, which divides the project management process into sprints to coordinate the product development process with the needs of the client (or customer). It is an iterative process where the sprints can last from 2 up to 4 weeks.

The theory on which this method is based is the one of empirical control of the processes (or empiricism), according to which, from one hand, the knowledge derives from the experience, and on the other hand, the decisions are based on what is known. For this reason, it involves an iterative process with an incremental approach that optimizes, step by step (and sprint after sprint), the predictability and the risk control.

Scrum is a method based on principles such as transparency, inspection and adaption. The main components are divided under three subcategories: roles, artefacts and events.

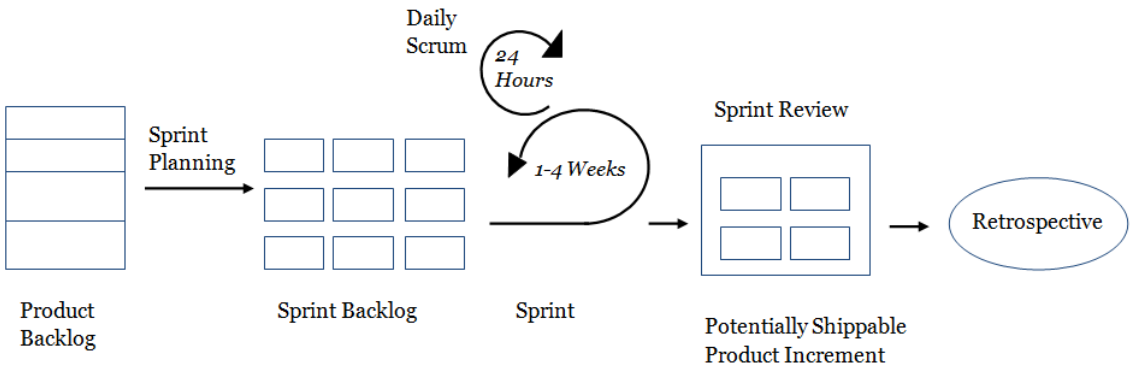

Figure 14: Scrum at a glance

5.3.1. Scrum roles

There are three roles defined within the Scrum Team, which are working in close connection to guarantee a continuous and fast flow of information. They are Scrum master, product owner and the development team.

Scrum master is the process manager, who is responsible for the most important task inside the team, which is to ensure that the Scrum methodology is being successfully comprehended and executed. He has to ensure that the team works coherently with the development of the project, by eliminating any external obstacles that can compromise the productivity of the team and organizing comparison meetings.

The product owner is the one that knows all the requirements of the product and carries out the interests of all stakeholders. He is the interface between the business, the clients and requirements of the product on one side, and the team on the other side. Moreover, the product owner has to maximize the value of the product and the work done by the development team.

The development team is the group of cross-functional and self-organized professionals, whose number is somewhere between 5 and 9 members. The team is responsible for the development of the product, for the functionality testing and the organization of priorities by transforming them in tasks to complete to end the particular sprint.

5.3.2. Scrum artefacts

The number of artefacts is three, the same as the number of roles. The artefacts are designed to maximize: the transparency of the key information, for the Scrum team, as well for all the stakeholders; and the opportunity for inspection and adaptation. The artefacts are the following: product backlog, sprint backlog and increment.

The product backlog is the document that contains the list of all the compulsory requirements for the realization of the product. The product owner

is responsible for its content, its availability and ordering of its elements based on the priority of execution.

Spring backlog is the document that defines all the tasks to be completed in each sprint. It is an estimation made by the development team concerning the priorities indicated in the product backlog and to the necessary work for achieving the goals of the sprint.

The increment is the sum of all the elements of the product backlog completed during a sprint and the previous sprint. At the end of the sprint, the increment has to be made based on the agreement if the development team, all that to ensure a usable product.

5.3.3. Scrum events

There are four formal events, with a fixed duration, used in Scrum to create regularity, to synchronize the activities and to reduce at minimum the necessity if non-defined meetings. The objective of these events is to allow critical transparency and inspection of the progress of the project. The events are sprint planning, daily Scrum, sprint review and sprint retrospective.

Sprint planning is the reunion where the product owner has compiled the product backlog and, in the presence of the development team and Scrum master, describes the most important points and defines the objectives to be achieved in the next sprint. At the end of the meeting, the Scrum master can fill in the sprint backlog.

Daily Scrum is a short daily confrontation, which usually lasts 15 min, between the development team and Scrum master. On that daily encounter, the Scrum master notes the work done in the previous day and creates a plan for the following 24 hours, until the next daily Scrum, so that they predict and synchronize their activities,

The sprint review is a revision at the end of every sprint which values if the predefined goal is being reached and what are the results. In this one, along

with all the members of the Scrum team, also the client of the product is participating. The work done up to that sprint is shown to the client.

The sprint retrospective is an additional retrospective analysis performed with the participation of the entire Scrum team to evaluate what to continue doing, what to stop doing and what to improve in the next sprint to obtain even more efficient performance.

All that has been described in this chapter until this point answers the question of what are the advantages of applying Agile and Scrum method in a company. Shortly, in conclusion, if these methodologies arose in IT context, applied in the software product, the processes and the principles, which regulate them, are making them beneficial for any business system that finds itself facing the management of innovative and complex projects [25].

5.4. The sprint

In previous subchapters, the word sprint was often found, but it was not explained. Actually, the sprint is the heart of the Scrum, or better said, it is a time interval of a month, or something less, during which a “done”, usable and potentially releasable increment is being created. The sprints have a constant duration during the development of software. A new sprint starts immediately after the preceding sprint is being concluded. The sprints contain and consist of all four Scrum events, plus the developing work, and the order is the following: sprint planning, daily Scrums, development, sprint review and sprint retrospective.

There are some protocols to respect during a sprint, which are:

- No modifications are made that may compromise the goal of the sprint;
- No minimization of the quality goals;

- The application can be clarified and renegotiated between the product owner and the development team as more information is learned.

Every sprint is defined as a project with a horizon up to one calendar month, and it is used to accomplish the given task. Each sprint defines what has to be built, with a design and flexible plan for the building, the work and the completed product increment. The duration of a sprint is limited to one month, since longer the horizon, higher complexity, and consequently, higher the risk. Sprints allow predictability in the way that the progress toward the sprint goal is inspected and adjusted at least once every month. Also, the cost of the risk is limited to one month [30].

5.4.1. Cancellation of a sprint

A sprint can be cancelled before the deadline of its time interval. The cancellation can be executed only by the product owner since only he has the authority for such an action. However, the decision for cancellation may be influenced by the stakeholders, by the development team or by the Scrum master.

A sprint would be cancelled if the goal became obsolete, which means the cancellation occurs if the company changes direction or there is a change in the market and technology condition. Roughly speaking, the point of cancellation is reached when there is no sense of the existence of the sprint. Nevertheless, because of the short duration of the sprint, the cancellation rarely makes sense.

When the sprint is being cancelled, the completed elements and the final product backlog are examined. This is due to the fact that if part of the work has the potential to be released on the market, often it is accepted. All the other incomplete items in the order book are re-evaluated and re-entered in the same order book. The re-evaluation is done since the work done on these incomplete items is often undervalued.

However, the cancellations are extremely rare, because they are traumatic for the Scrum team in terms of lost work, lost time and consumption of resources.

5.4.2. Planning of a sprint

What is accomplished in a sprint is planned during the event sprint planning. The plan is created in collaboration with all the members of the Scrum team. The maximum time interval assigned to a sprint planning for a sprint with a duration of one month is eight hours. The duration of that interval is proportional to the duration of the sprint, which means, shorter the sprint, shorter the sprint planning. The Scrum master ensures that the event takes place and that the participants comprehend the goal. Another role for the Scrum master is to push and stimulate other members of the Scrum team to respect the deadlines.

Sprint planning gives the answers to two important questions, which are:

- What can be delivered in the increment from the upcoming sprint?
- How will be organized the necessary work for realizing the increment?

5.4.3. Sprint goal

The sprint goal is a fixed goal, an objective, for the sprint that can be achieved by implementing the product backlog. It provides a guide for the development team for building the increment. The sprint goal is created during the sprint planning meeting and to the development team, it offers flexibility about the implemented functionalities within the sprint.

The selected product backlog elements are offering a coherent function, which may be the sprint goal. Actually, the sprint goal can be any other

coherence that induces the development team to work together instead to work on separated initiatives. While the development team works, it should keep the sprint goal in mind.

Intending to meet the sprint goal, the Scrum team implements the functionality and technology. If the work is different than what was initially predetermined by the development team, then the development team, collaborating with the product owner, negotiates the sprint backlog scope within the sprint.

5.4.4. Daily Scrum

The daily Scrum is a 15 minutes long event that helps the development team to synchronise the activities and to create a plan for the following 24 hours. This is performed by inspecting the accomplished activities of the last daily Scrum and forecasting the work to be done for the next daily Scrum.

To minimize complexity, the daily Scrum takes place every day at the same time. Some of the activities that team members are doing during the meeting are: explaining their work to other members; describing their intention per achieving the sprint goal for that day; and exposing any obstacles that may compromise the sprint goal.

The development team uses the daily Scrum for inspecting the progress made towards the sprint goal and to verify if that progress is in accordance with the sprint backlog. Such a short activity improves the probability for the Scrum team to achieve the sprint goal.

Every day, the development team should evaluate how to operate as a self-organising team to achieve the sprint goal and to create an increment by the deadline of the sprint. The meetings of the development team, and/or other members of the Scrum team, are not finished with the daily Scrum. Instead, after the daily Scrum, there are reunions for detailed discussion o planning and re-planning the remaining work.

The role of the Scrum master is to ensure that the team will meet daily, but he is not responsible for what is conducted at that meeting. He has to push the development team to maintain every day the event and not exceed the 15 minutes time interval.

There are many advantages of the daily Scrum meetings. These meetings are making progress in the communication between the team members by increasing the level of knowledge of the team, identifying the barriers to be removed and promoting the rapid decision process. This is a key to inspect and adapt to the meeting.

5.4.5. Sprint review

At the end of the sprint, a sprint review is held to inspect the increment and, if necessary, to adapt the product backlog. During the review event, the Scrum team and the stakeholders are collaborating on what was done in the sprint. Based on this and eventual changes made at the product backlog, the participants are working on the next steps for optimization of the value of the results of the sprint review.

The sprint review is an informal meeting with a goal to arouse feedback and encourage collaboration. It is a four-hour meeting for a month-long sprint. For shorter sprints, the time window is shorter, too. Similarly to the daily Scrum, the role of the Scrum master is to ensure that the team is participating in the meeting and the members are respecting the goal and the given deadlines.

The timeline of the sprint review includes the following elements:

- The product owner invites the Scrum team members and main stakeholders;
- The product owner explains the performed elements from the product backlog, and what did not end well;

- The development team discusses the flow of the sprint, the problems and obstacles that occurred, and how they were resolved;
- The development team shows the accomplished work and their solution for the increment;
- The product owner presents the product backlog with the expected target time;
- The entire group collaborates on what to do next so that the sprint review offers a valuable involvement in the succeeding sprint planning;
- A review on what are the next moves based on how the market or the potential use of the product may have changed;
- A review on the chronology, the budget, the capacities and the market for the anticipated version of the product.

The result of the sprint review is a revised product backlog for the potential items of the next sprint's product backlog. Nonetheless, the product backlog may subject to modification to satisfy new opportunities.

5.4.5. Sprint retrospective

A sprint retrospective is an event in between sprint review and next sprint planning, and it represents another opportunity for the Scrum team to check what can be improved in the subsequent sprints. The time dedicated for this event is three hours (in case of a one-month-long sprint), and similarly to other events, it depends on the duration of the sprint, which is shorter the sprint, shorter the sprint retrospective. Moreover, the roles of the Scrum master are the same as before: ensuring the event is accomplished, guarantying participation of the members and understanding the purpose of the event.

The aim of the sprint retrospective is:

- To inspect the performance of the last sprint in terms of human resources, relations, processes and instruments;

- Identification and order of the main well-end items and eventual improvement;
- Creation of a plan to implement the improvements in the way of operation of the Scrum team.

One of the roles of the Scrum master is to encourage the Scrum team to improve, in the sphere of the Scrum process, its development process and the practices to make it more efficient for the succeeding sprint. During each sprint retrospective, the Scrum team plans ways to increase the quality of the product by implementing the definition of "done" as appropriate. By the end of the sprint retrospective, the Scrum team should identify the enhancements to be implemented in the next sprint.

The implementation of these enhancements is the adaptation to the inspection of the Scrum team itself. Although the improvements may be carried out at any time, the sprint retrospective offers a formal circumstance to dedicate to inspection and adaptation

5.5. Kanban

Kanban is a method for managing the development work with a particular accent to "just in time" delivery without overloading the team members. The development process in this methodology is transparent along the entire time, since the definition of the activity, until the delivery to the customer. In this way, all the participants can see and follow the development.

The Kanban method is an approach to the incremental evolutionary processes and the system changes of the organizations. It uses a limited pull system while working as the main mechanism for exposing the problems in the system or the process, and to stimulate the collaboration for continuous system improvements. Visualisation is an important aspect in Kanban because with the aim of visualisation the work and the flow of the work are better understood.

There are four basic principles that take part in the Kanban methodology:

1. It starts with the existent process: the Kanban method doesn't recommend a specific set of roles or process phases. That is the reason why it starts with existent processes and stimulates continuous, incremental and evolutionary changes in the system.
2. It agrees to follow the evolutionary change: the organization and the team should agree on the fact that the continuous, incremental and evolutionary change is to make improvements on the system and to make them remain more cohesive for evolved changes.
3. Respect the current process, roles, responsibilities and titles: the organization likely has some elements working acceptably and they are worth to be preserved. The Kanban method tries to dispel the fear with the aim to facilitate future modifications. It tries to eliminate the initial fears by accepting to respect the current roles, responsibilities and working titles to expand the support.
4. Leadership at all levels: acts of leadership at all organization levels are encouraged, from the individual contributors to the senior management. Traditionally, Kanban was a physical blackboard, with magnets, plastic buttons and sticky notes on a blackboard to represent the working items.

However, in the last years, more and more project management software tools have created Kanban boards online. A Kanban board, regardless if it is physical or online, is made up of several lines or columns. The simplest boards have three columns: what to do, what is doing and what is done. The columns for a software development project can be columns of analysis, development, test, approval and distribution.

The Kanban cards, similar to the sticky notes, represent the work and each card is positioned on the board, in the lane which represents the status of that work. These cards communicate the status at a glance. Moreover, cards in different colours may be used for detailed representation.

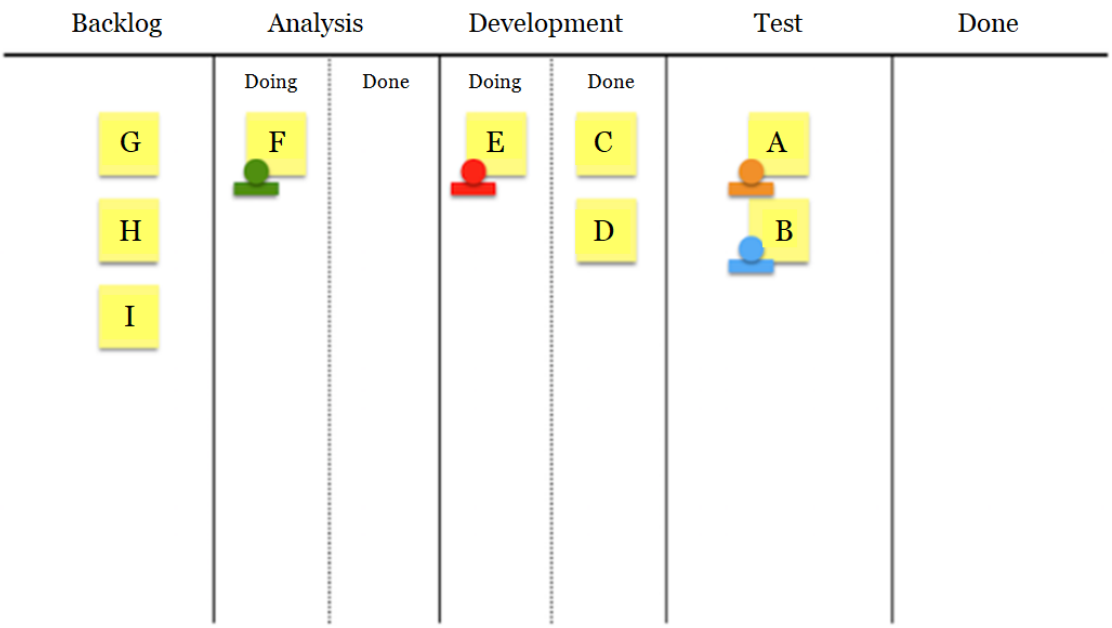

Figure 15: An example of Kanban board

The Kanban method allows the incremental improvement of the process through the repeated discovery of the problems that influence the performance of the process. This method supports the gradual and continuous improvement towards the higher performances and better quality. Moreover, the Kanban method prefers evolution rather than the revolution of the process.

The Kanban principles that help the software development team to provide a better product are:

- Optimization of the existent processes: The introduction of visualisation and the limitation of the work-in-progress (WIP) catalyze the change with minimal interruption;
- Deliver with better quality: Limiting the work in progress and defining the policies for the definition of the working priority can bring higher attention to the quality. The policies can also confront directly the quality criteria;
- Improving the predictability in the delivery times: There is a correlation between the amount of the work in progress, the

delivery times and the defect percentage. Limiting the WIP makes delivery times reliable and keeps low the defect rates;

- Improvement of the employers' satisfaction: Kanban reduces the change of context and pushes the work at speed with which the team can complete it. Working with a more uniform and predictable rhythm means that the employees are never overloaded;
- Providing break (truce) to ease the improvement: The creation of temporal truce spaces in work chain improves the reactivation of the urgent requirements and predisposes the improvement of the processes and consequently, the improvement of the quality;
- Simplification of the definition of priorities: Kanban allows rapid reorganization of the priorities in order to adapt to the market changes;
- Ensure the transparency on the design and functioning of the system: Better visibility creates trust with customers and the manager. Moreover, it shows the effects of actions and inactions. Consequently, the collaboration improves.
- It enables the appearance of "high maturity" organization: As the improvements are being implemented, the organizational maturity improves leading to effective decisional process and consolidated risk management. The risk, managed in an appropriated way, gives predictable results.

5.6. Other Agile methods

In the previous subsections of this chapter, there were introduced different methods to produce software production in the Agile method and their potential impact on the software quality. The attention is concentrated on the methods that offer complete frameworks with processes and practices more or less tangible, which cover the software development process as much as possible.

The Agile approach, however, recently brought a larger amount of researches and many new interesting ideas (e.g. lean software development, development based on functionality, open-source software development, adaptive software development etc).

5.7. Refactoring and review of the code

The refactoring is a maintenance activity where the source code is restructured to improve the quality while the extern behaviour of the system is preserved. The term “code review” may refer to a series of activities, from simple code reading to a 20 people-meeting where the code is analyzed line by line.

The refactoring process often is defined as a reconstruction process for the source code in order to reduce at minimum the error or the defect, both as correction act and as prevention. For example, the code clones and other redundancies in a program are considered a threat to the further development of the software. These errors in the source code are noted as code “stinks”. Another advantage of refactoring, beyond the improvement of the program quality, is that of improving the structure of the software in the way it becomes easier for understanding and comprehending.

Advantages of the software refactoring are:

- It allows to remove the duplicated code and other “stinks”;
- It enables to improve the quality of the software design;
- It increases the comprehensibility of the code;
- It decreases the time for project evolution, particularly in the management of the source code.

Members of a code review team are the author and the reviser. The author is the one who writes the code and sends it for review, while the reviser is the one who reads the code and decides if it is ready to be added to the base, to be united with other code. One review can have more revisers.

Before the start of the program review, the author should create a list of changes. This is a series of source code changes that the author desires to unite it to the base of the team code. The review starts when the author sends the list with changes to the reviser. The code reviews occur in cycles, where each cycle is a two-way event completed between the author and the reviser. The author sends the modifications and the reviser gives feedback to those changes.

Each review has one or more cycles and it ends when the reviser approves the changes. The author likely interprets the critics as an implication for being an incompetent programmer. The code reviews are an opportunity to share knowledge and to make known engineering decisions. But it cannot happen if the author perceives the discussion as a personal attack.

Some refactoring scenarios were found to improve some qualitative aspects of the software, but to weaken others. These results lead to the conclusion that the refactoring doesn't always improve all the software quality aspects. The developers confirm that refactoring is a "code transformation that improves some aspects of the program behaviour like readability, maintainability or performance".

The refactoring asks a multidimensional valuation. Therefore, it is convenient to consider wrong limiting the study of the impact of a given refactoring scenario on quality to a certain quality attribute, then obtaining some negative results and therefore declaring general conclusion that the given refactoring scenario weakens the quality of the software.

5.8. Pair programming

Pair programming is an Agile software development technique where two developers share a single workstation. In this technique, one of the developers is the driver, the one who controls the mouse and the keyboard to write the code, while the other is considered as navigator, reviewing the written code and giving tactical and analytical feedback. This pair changes the roles at

regular intervals, giving reciprocally the same possibilities for operating the directing the work.

The final goal at the end of pair programming is to provide a medium for obtaining a better quality of the software, offering at the same time many secondary advantages that improve the capacity of a team to continue providing useful features to the customers.

Even though code reviews often find typos and simple errors, they don't provide the same level of understanding of the problems related to the quality of the design and the architecture of the software.

Some of the important pros of the pair programming are:

- Many errors are caught while typed instead with the quality control tests or on the field (due to continuous reviews);
- The content of the final defect is statistically lower (due to continuous reviews);
- The design is better and the length of the code is shorter (due to simultaneous brainstorming and couple forwarding);
- The team resolves faster the problems (due to couple forwarding).
- People learn a lot more about the system and the software development (due to learning in line of sight);
- The project ends with more people understanding each part of the system;
- People learn to work together and to talk more often together providing better information flow and group dynamics;
- People love more their job. The development cost for these benefits is not 100% predictable, but 15%. This is compensated with shorter and cheaper tests, guaranteed quality and on-field support [28].

Chapter 6 – DevOps methodology

DevOps is a software development methodology that emphasises collaboration and communication between software developers and IT operations professionals [31]. DevOps can provide competitive advantages to the companies by helping to faster deliver better software, enabling continuous innovation; in addition to ensuring quality, safety; thus greater reliability to the produced software [32].

Therefore, by adopting the Agile methodologies, DevOps has the goal to create culture and environment where the software design, testing and release can take place in a rapid, frequent and efficient way. With a better workflow, it can offer to the organizations the flexibility to change faster, without sacrificing the values and strengths for which this methodology was born [31].

In traditional organisations, the functions “development”, that is the developers, and “operations”, that are the IT professionals and systems engineers, are distinct. By simplifying, it is possible to affirm that the first ones are working on software development, while the second ones on the release in production and in the correct functioning of what has been released.

Since certain functions have different goals, paradoxically, they risk entering in conflict. The developers aim to release quickly new and improved features, and thus they would be able to deliver software every day; while the “operations” part, are pointing to have always functioning and efficient system, therefore, they tend to maintain the thing in their current (working) status as long as possible.

This difference slows the releases, and thus the business. DevOps is a combination of the two terminologies, “development” and “operations”. This new terminology refers to a software development methodology that maximizes the collaboration, the communication and the integration between the software

developers and the IT professionals (sysadmins, DBA, etc). The purpose of this methodology is to make an organisation able to products and software services quickly, avoiding the aforementioned “conflicts”.

As a part of an organisation that provides a “hybrid” service, as an example for DevOps may be close collaboration between the developers and the verifiers/valuators, to provide a predisposition for a suite of an automatic and manual test to guarantee and verify what has been developed; and between developers and system engineers with the aim to prepare an automatic script which allows the release of the developed software in different environments [33].

The movement DevOps has produced, and continues to release, numerous principles that can be adopted by organisations of all dimensions (Figure 16 [36]). All of these principles brought up an approach with a goal to improve the way in which the business provides value to its clients, suppliers and partners [32].

The DevOps assumes that in the company, there is an inter-functional team, where every resource is responsible for everything. Today, in the business environment, it is still very rare to find an organisation that can qualify itself as DevOps completely [34].

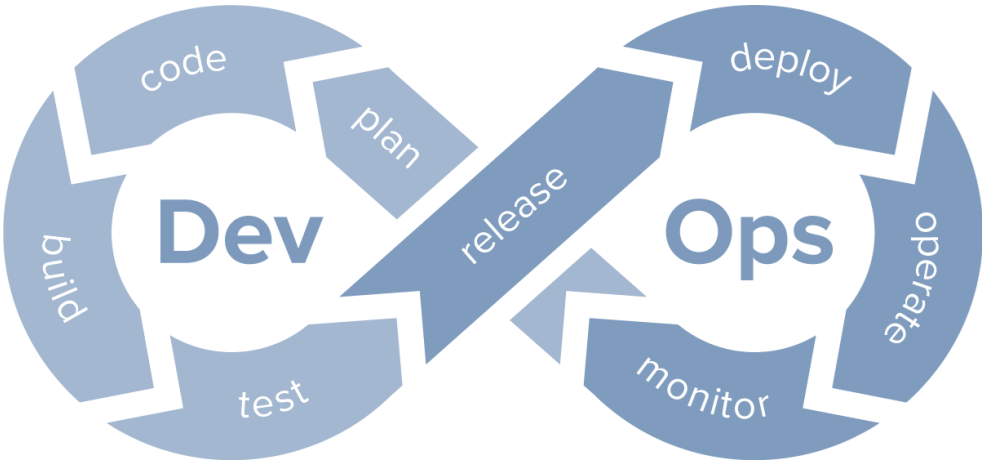

Figure 16: DevOps methodology

There are eight advantages with the adoption of DevOps methodology, which are the following:

- The quality improvement of codes, products and services (lower number of faults, greater success rate of changes, etc);
- Increase in effectiveness (e.g. greater time spent in activities to create additional value, greater additional value per client, etc)
- Improvement of the time to market;
- Better IT alignment and business reactivation;
- Faster, smaller and more frequent releases;
- Less waste and fewer anomalies;
- Improvement in productivity, clients satisfaction, staff satisfaction;
- Lower long-term costs [32].

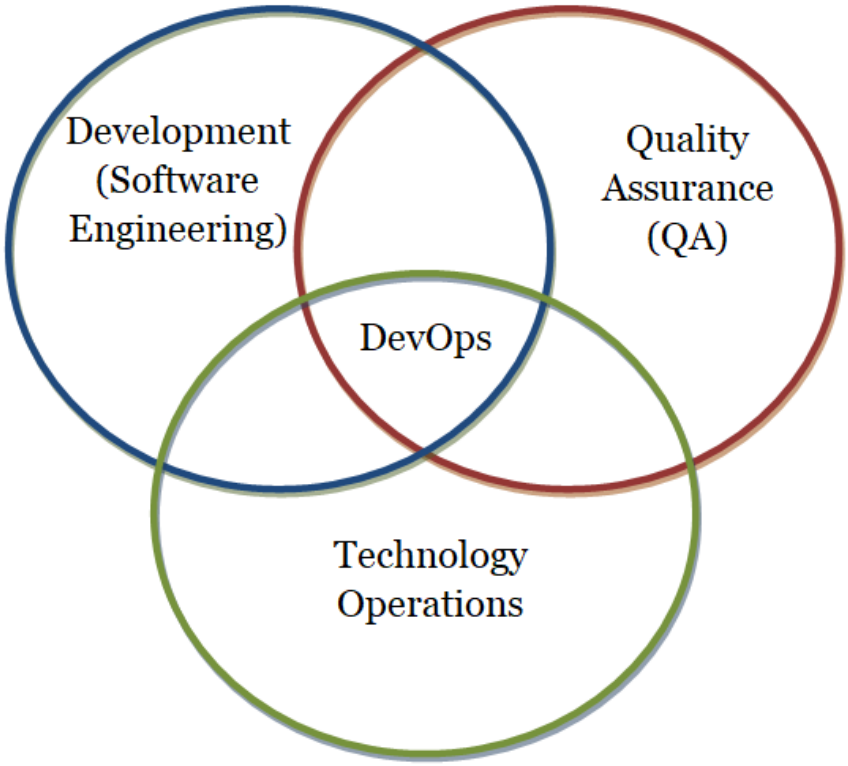

Figure 17: Collaboration between departments

Beyond the Agile approach, fundamental characteristics of the DevOps development system are the continuous integration CI and continuous development CD. Continuous integration means that in the development process the tests of the code piece are continuous and automatic, while the continuous development wants to affirm that the process of putting the validated code into production after the proper testing becomes automatic.

All of this contributes to speed significantly the software releases times.

In the past, certain companies managed the production of the new code managed the putting into production of the new code at predefined times. But, nowadays, given the speed required by the customers, this model for release cycles results rather obsolete and in complete antithesis to the DevOps which points precisely, thanks to the automation, to make the new release available as soon as possible.

Another fundamental aspect of the DevOps is the infrastructures as codes (IaC) that is the generation of the automatic methods for configuration and implementation of the infrastructure. Specifically, having infrastructure as code means that it can be incorporated into the other DevOps processes such as the testing phases and the start of the production.

Different business realities already memorize all of their infrastructure configurations in the GitHub repository, and these are tested and distributed continuously exactly like the rest of the continuous integration and implementation processes, processed dozens of times a day. Dealing with software code or infrastructure, everything passes to the same automatic process methodology, regardless of what is modified.

The DevOps have a fundamental role, especially in the cloud. The companies with cloud infrastructure need to manage constantly a series of resources that can be incorporated by DevOps.

Moreover, and above all, on the cloud it is possible to automate the application releases and updates and obviously, to create infrastructures as code, easily replicated in different environments and immediately ready for use.

In the cloud, the DevOps team can create applications written especially for the cloud (cloud-native), to manage and orchestrate containers to simplify and streamline processes and to make the applications' "time to market" with fast and continuous publications and releases [34].

6.1. Peculiar characteristics of DevOps

6.1.1. Collaboration and trust

The culture is a success factor n.1 for the DevOps. To build a culture of shared responsibility, transparency and faster feedback is the basis of each high-performance DevOps team. Lack of visibility and shared goals means lack of dependence planning, misaligned priorities and mentality of type "not my problem", with consequent slowness and releases with inferior quality.

DevOps is that change of mentality which leads to watching the development process holistically and breaking down the barrier between the Dev and Ops.

6.1.2. Release faster and work smarter

The velocity is everything. The teams who practice DevOps release software products more frequently, with better quality and stability. The lack of automatic tests and review cycles block the release into production, resulting in long error resolution times, thus reducing the velocity and the confidence of the team. Different instruments and processes increase the OPEX (operating expenses), determining the context change and slowing the dynamics. Through the automation with standardized tools and processes, the teams can increase the productivity and release products more frequently with fewer hitches.

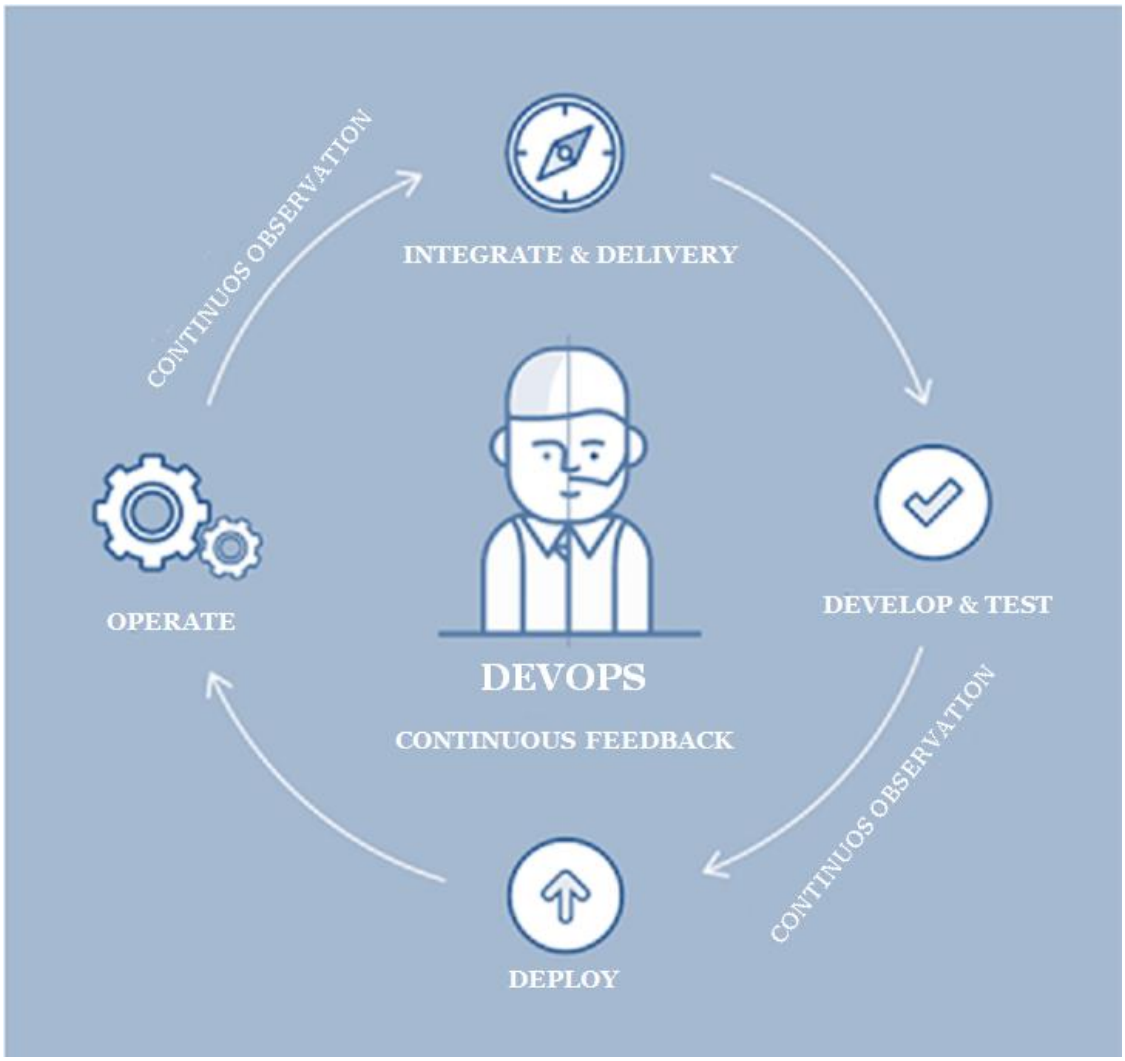

Figure 18: DevOps – Continuous feedback

6.1.3. Acceleration of resolution time

The team with a faster feedback cycle is the winning team. The full transparency and the communication without interruptions give the possibility to the DevOps team to reduce at minimum the inactivity and to resolve the problems faster than usual. If the critical problems are not resolved quickly, the client's satisfaction disappears. The absence of open communication increases the tension and the frustration between the teams Dev and Ops, while direct communication allows them to correct the errors and the problems more easily and to unblock the pipeline more quickly, or the result of release products.

6.1.4. Better management of unplanned work

The unplanned work is a reality which each team deals with and that often influence productivity. With established processes and clear priority definition, the teams Dev and Ops can manage better the unplanned work, while continuing to concentrate on the planned work.

The transition and the priority assignment of unplanned work between the teams and different systems are inefficient and distract from the work in progress. Nevertheless, through augmented visibility and proactive retrospective, the teams can be more careful and can share the unplanned work.

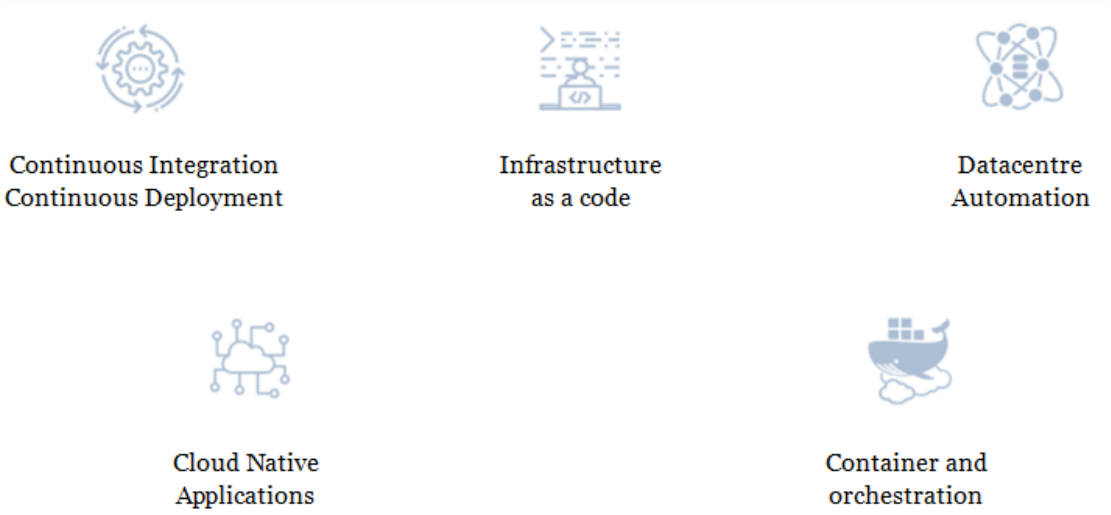

Figure 19: Advantages from the utilization of DevOps

The advantages of the usage of DevOps in a company or organisation are illustrated in Figure 19 [35].

Chapter 7 – Model-based Development

7.1. Introduction to MBD

The complexity of the embedded system (along with the ever-increasing number of embedded systems) demands solutions that are economically and time-efficient. In other words, the market is asked to reduce the production time and the cost for the development of such a system, having in mind that the size of these systems and their parts (processors and microchips) keeps shrinking [38].

In order to respond to the requirements of the market, another methodology has been adopted. Model-based Development (MBD), found also as model-based software design or model-based design, is a framework that performs verification and validation of the product and it is applied in the virtual prototyping of embedded software. The MBD follows the software product from the initial steps, analysis and design, up to the final verification and validation testing, by including different disciplines, functionality, efficiency and cost/performance optimization. "Modelling" in the design/development process, by definition, is a mathematical description of the physical system. It means the dynamics of the system and the subsystems are represented with formulas and graphically for better comprehension and simplicity [39].

Traditionally, in the design part of the process, the information for the design of the product was in text-based form, thus with reduced useful information for the software engineers. Opposite of it, the MBD can cover a greater amount of details compared to the traditional way, diminish the errors and misunderstanding which can lead to non-functional products. Moreover, the model-based development replies to the market challenge for compact, durable, life-long maintainable, customizable and recyclable products by

allowing the various real-time application. So, the developers are authorized to forecast the performance of the product, can test the system functionality under different inputs and, additionally, the test is performed in a simulation environment. Being modelling one of the most essential parts of the design (and of the overall) process, the efficiency of this methodology is notable, since, as the name suggests, this design and development is based on the model.

After this step is completed, with a built model, the second step is coding. Instead of the manual coding, code is automatically generated and the software product is simulated. This step not only improves quality but saves time and it is cost-efficient. Then by rapid virtual prototyping, the behaviour of the whole system (mechanical, electrical, embedded software) is studied, to check if it satisfies the predefined requirements.

As a conclusion from what has been stated before, one of the reasons for the companies to adopt model-based development is its potential: it implements a single design platform to optimize overall system design.

The importance of the MBD is observable by its application. It is utilized in the design of highly complex systems, such as autopilots, guidance systems, medical devices, different electronic control modules in the vehicles, such as ABS and other design companies that appreciate the advantages which it offers [37].

7.2. MBD framework

The framework (Figure 20 [37]) of the model-based development includes six steps that can be displayed into the V-diagram lifecycle model, which has been explained in chapter 3. The steps are the following:

- Modelling;
- Simulation;
- Rapid prototyping;
- Embedded deployment;

- In-the-loop testing;
- Integral activities [37][38].

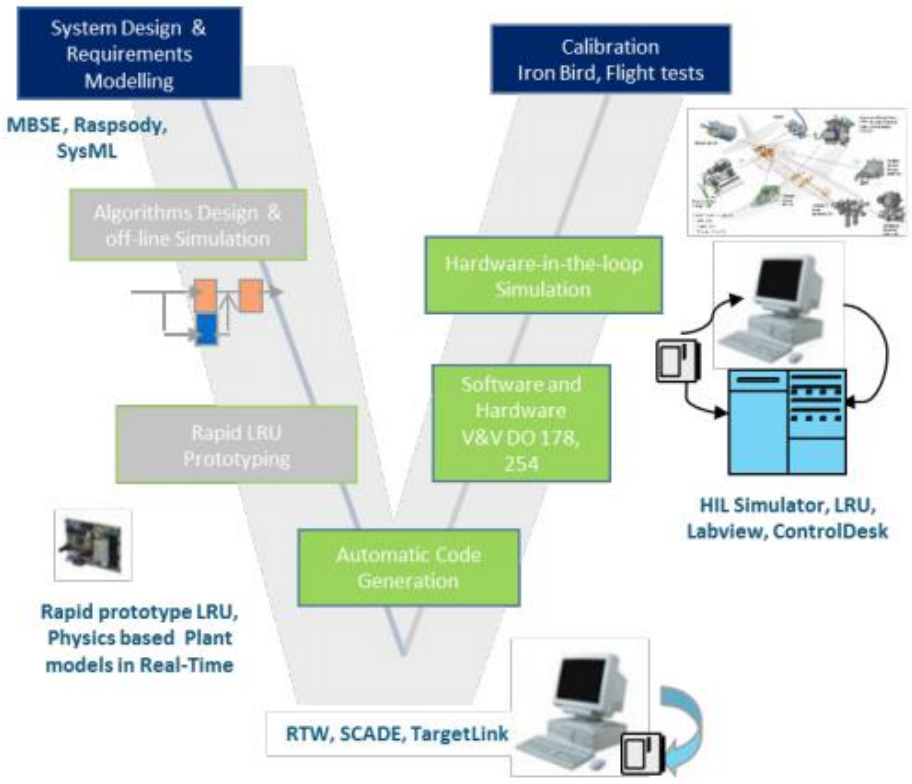

Figure 20: MBD workflow within V cycle

7.2.1. Modelling

The first activity, system modelling, is where the model of the given system is built, which means a mathematical and behavioural representation of that system is created. The models are dynamic so the behaviour of the system can be anticipated at any time knowing the inputs and the current state.

The system, along with the equations, can be modelled graphically with block diagrams and lines. With this step, even the complex control and communication systems can be visualised and better understood.

There are two types of models depending on the dynamics. In other words, if the model contains continuous dynamics it is defined as continuous-time system, while in case of discrete dynamics, the model is classified as a discrete-time system.

Continuous-time systems are changing their state continuously over time (time flows without any break or interruption) and they are an illustration for real-world effects and analogue signal. As examples of continuous-time systems are: vehicle, chemical reactor, turbine, actuator etc. On the other hand, the discrete-time systems change their state at finite intervals of time. An example of this model is embedded software running on a microcontroller. To start with the execution of the algorithm, this software relies on internal clocks. The best model is a hybrid one that contains both types of dynamics.

7.2.2. Simulation

The second step in the MBD framework is the simulation where the model is tested. What is the hardware prototyping for the hardware that is the simulation for the software. During this step, continuous-time systems are solved using numerical integration. For this purpose, there are two types of solvers based on the way they calculated the step: fixed-step and variable-step solvers.

The difference between the two solvers is the way how they compute the next continuous state. As the name suggests, fixed-step solvers are computing at fixed periodic intervals of times, while for the variable-step solvers it is at non-periodical intervals of time. Another difference is the method they use, therefore fixed-step solvers use only explicit methods, while the variable-step solvers use explicit or implicit methods. Explicit method is calculating the current state based on the previous state, while the implicit method is calculating the current state based on both the previous and the current state.

The objective of the simulation is to determine a precise approximation of the behaviour of the system by choosing suited sample time and integration

method. The sample time for the fixed-step solver is the fixed step time, while for the variable-step solvers it is the maximum time of the allowable sample times.

The choice of the solver is up to the software developers and depends on the dynamic of the model, but also on how they plan to deploy the model. Variable-step solvers are difficult to map to the real-time clocks and that is the reason why if the model is used to generate code, the best choice for the simulation is a fixed-step solver.

7.2.3. Rapid prototyping

The following activity within the MBD framework is the rapid prototyping of a product. For the engineers, it is a fast and cost-effective way for verification of the design at an early stage and eventual changes or trade-offs.

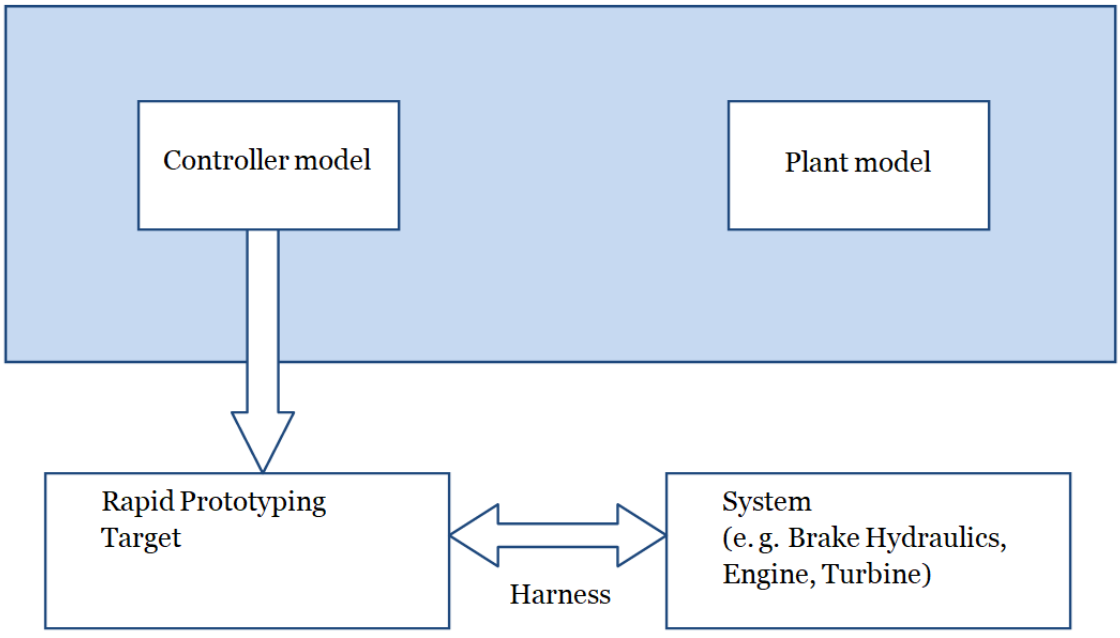

Figure 21: Bypass rapid prototyping

There are two types of rapid prototyping: bypass rapid prototyping (Figure 21 [37]) and on-target rapid prototyping (Figure 22 [37]). For both of

them, the code is generated only for the controller of the model. In the case of bypass prototyping, after the initial step, the code is cross-compiled and downloaded to a high-speed, floating-point, rapid prototyping computer where it is executed in real-time. On the other hand, for the on-target prototyping, the code is cross-compiled and downloaded on the embedded microprocessor inside the electronic control unit (ECU). That ECU, which uses a fixed-point integer processor, needs a detailed fixed-point model, opposite to the floating one utilized for the bypass prototyping.

I/O, for the first type of prototyping, is ruled by memory pod or emulation device connected to the rapid prototyping and also to an embedded controller, which is usually ECU in a vehicle. For the on-target prototyping, the I/O is managed by standard ECU devices. Then, in the case of both types of prototyping, the minor improvements are made “on the fly” during testing. The prototyping is successful when the performance requirements are met.

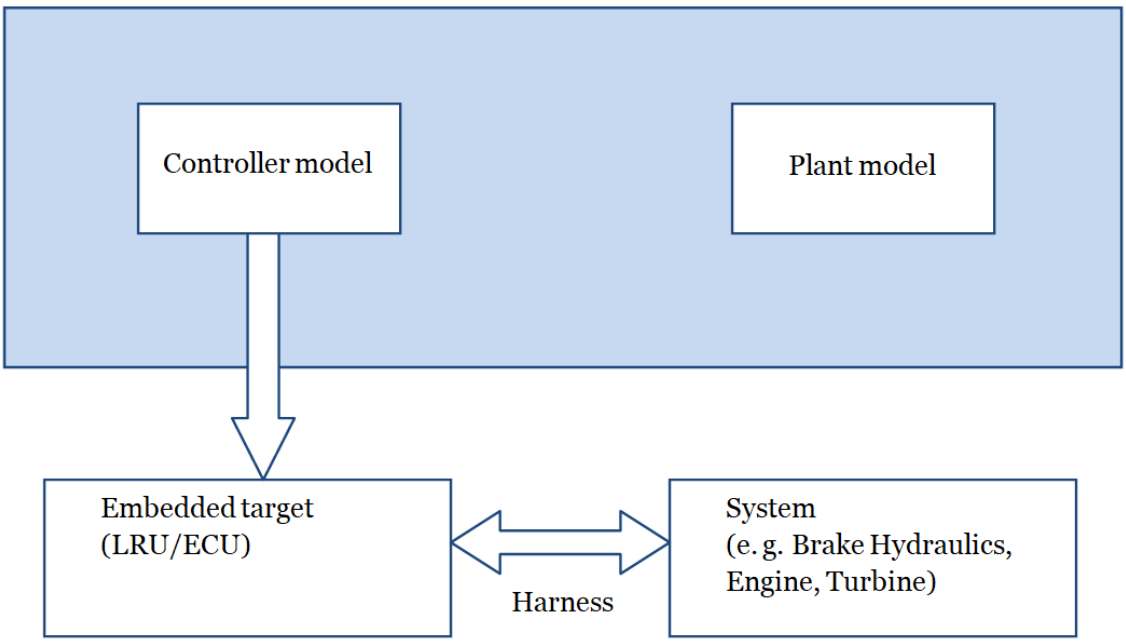

Figure 22: On-target rapid prototyping

7.2.4 Embedded deployment

The next step, following the rapid prototyping, is the embedded deployment. It is the step in which the controller model is transformed into fully executable software. The software developers have a task to create a suitable model, from the controller model, to be inserted on the embedded system hardware. Then, the optimized embedded code generated from the model is downloaded to the embedded hardware (microprocessor or ECU). The goal in this part of the MBD framework is to guarantee that the automatically generated code of the final product is fully integrated with the existing legacy code, I/O drivers and real-time operating system.

7.2.5. In-the-loop testing

In-the-loop testing is a step of the MBD framework that is performed after the completion of the embedded deployment. The V-model incorporates three feedback loops that are in the range from the requirements definition to the validation of the software on the hardware, e.g. validation on the vehicle. These loops are SIL, PIL and HIL, abbreviated accordingly from software in the loop, processor in the loop and hardware in the loop. The objective is to reduce the development time, to reduce the cost, to test the system at different stages, to complete the V&V process from the initial levels up to the final ones. All of this is accomplished to check whether the product meets the defined design requirements and satisfies the customer.

Software-in-the-loop (Figure 23 [37]) is the type of testing where the production code for the controller, generated from the model, is executed in non-real-time with the plant model. It also includes interaction with the user. There is a code wrapper that provides an interface between the generated code and the simulation and the code is executed in the host platform that has already been used in modelling part of the MBD framework.

Processor-in-the-loop (Figure 24 [37]) is very similar to the SIL since it is also a non-real-time simulation and the production code for the controller is

executed. The difference is the target where the code is executed, which in the case for PIL is the actual embedded processor. In this testing, the plant model is executing in the modelling environment, while the code is executing on the real processor and the communication between them is via CAN bus and real I/O devices. The objective of PIL is to check the behaviour of the generated code on the actual embedded processor.

Hardware-in-the-loop (Figure 25 [37]) is the testing where generated code for the plant model is executed on a deterministic, real-time computer. This testing includes one or more electronic control units, sensors and actuators, which are real components in a closed-loop with components simulated in real-time, such as the controller and plant model. In order to properly stimulate the sensors and receive actuator commands (inputs and outputs), power electronics are needed and signal conditioning is used. HIL is the final test in a laboratory before system integration starts [39].

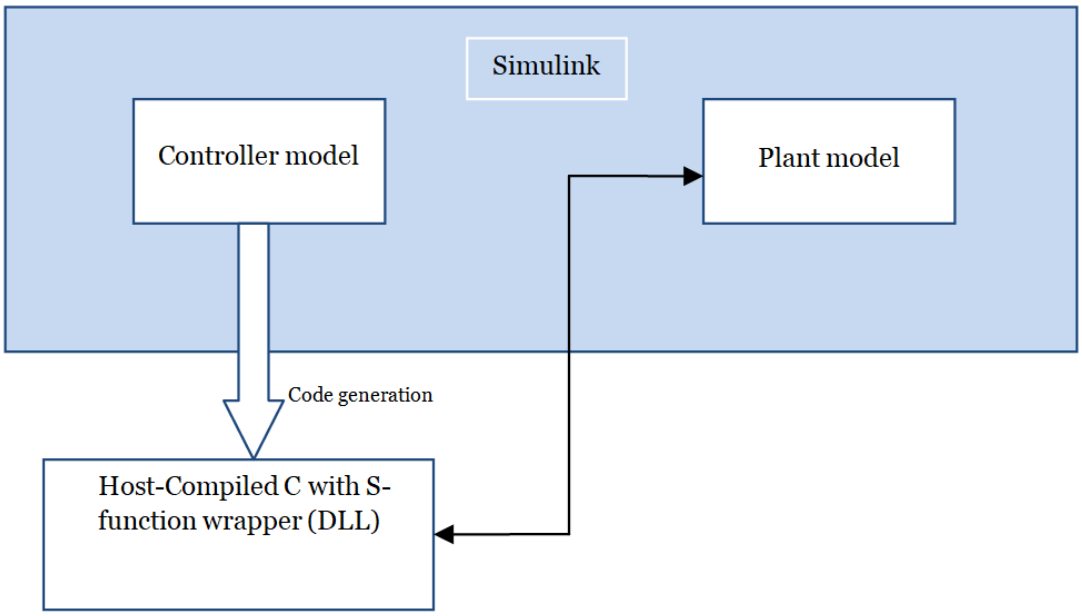

Figure 23: Software-in-the-loop (SIL)

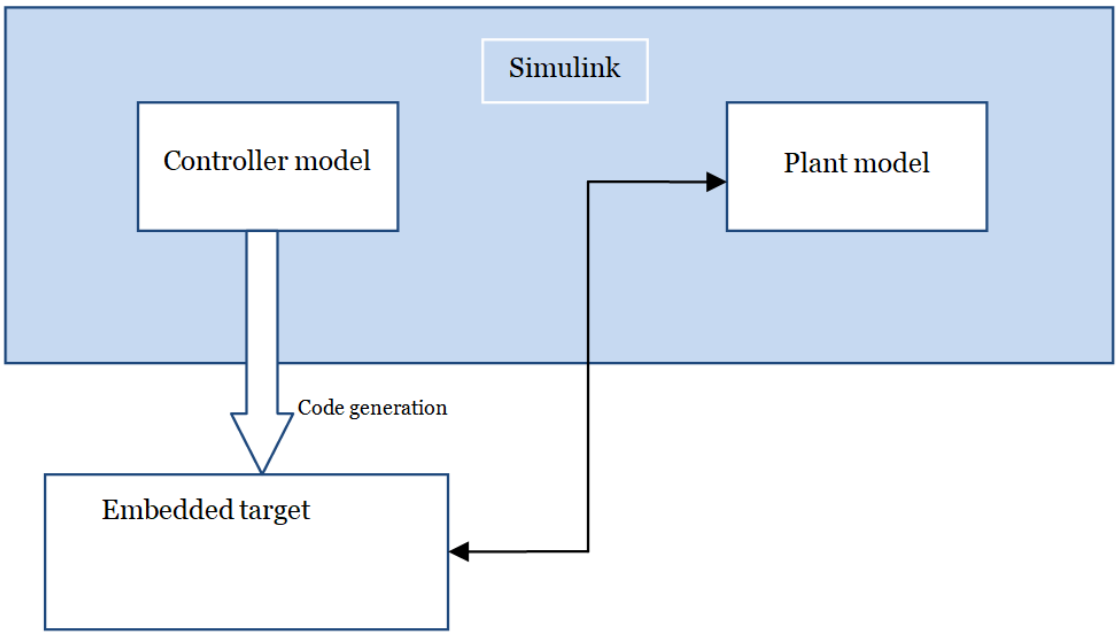

Figure 24: Processor-in-the-loop (PIL)

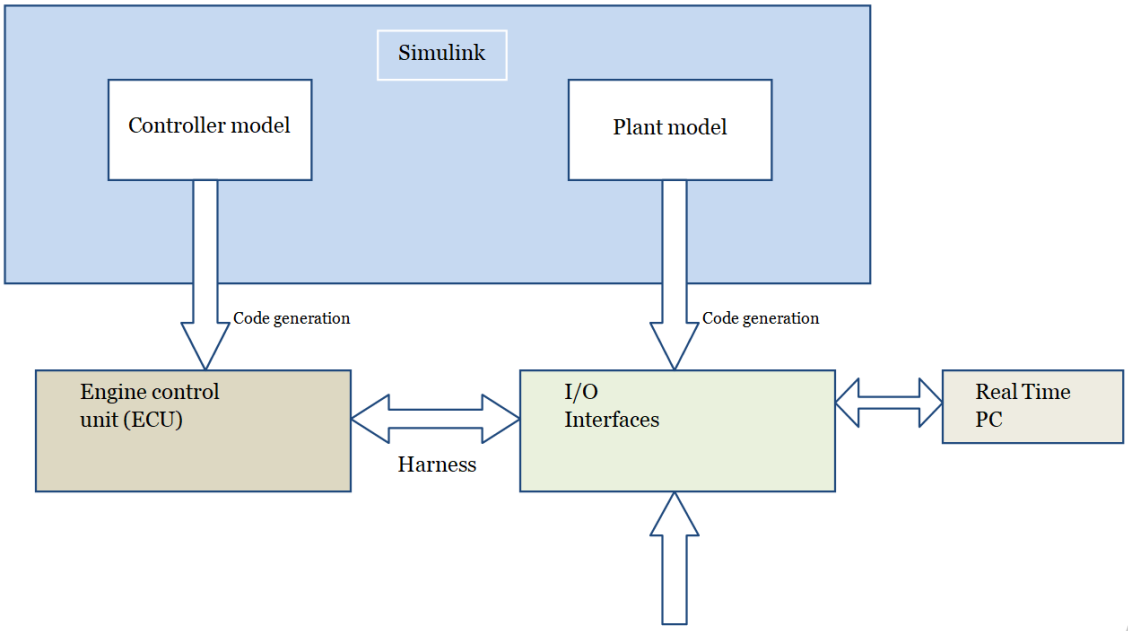

Figure 25: Hardware-in-the-loop (HIL)

7.2.6. Integral activities

The integral activities performed within the MBD framework are documentation, requirements traceability, etc. Even though they do not include any design or testing, they are important for completing the framework, to integrate the other steps. Therefore, the documentation is used to specify the content of each section, useful for both the developers and users. The traceability allows the software developers (or software engineers) to trace the defined requirements back to the generated code from the model by using interfaces between the model and requirements management source. The goal of the interface is to let the developers check the models and to document the changes they made [37].

7.3. Integration of MBD with Agile (Scrum) and DevOps; Application to the IVECO vehicles

The ECUs, electronic control units, are the most important part of the automotive system. They are hybrid components of the vehicles which control the electrical systems (and subsystems) in a vehicle. By hybrid, it means that they are made of mechanical and electronic parts.

From what has been researched and written in the previous chapters, the Agile method allows the companies to be flexible and to succeed in responding to the uncertain and changing environment. It is a reliable method, therefore suggested by the software engineers. On the other hand, Scrum is the framework used within the Agile method to provide effective team collaboration on complex products. Another method that was described in the previous chapters is the DevOps, which concentrates on the communication, collaboration and integration among the software developers for rapid product deployment. And the present chapter focuses on Model-based development in environments such as Matlab and Simulink.

To respond to the requirements of CNH Industrial (and IVECO S.p.A. as part of it, regarding the improvement of the IVECO vehicles) that were proposed in the introduction part of the present thesis, it is suggested to use a combination of Agile, Scrum, DevOps and MBD.

The integration between the vehicle (mechanic) and the control unit (electronic) is already robust. Since the software (informatics) is the one that pilots the ECUs, the reliability of the vehicular system increases, as well as the robustness and durability. From the performed analysis it is expected that the proposed methodology will aim to provide better overall performance and improved productivity.

By utilizing the Agile + Scrum, and managing it with Matlab + Simulink, a functioning model is created. Through these three, the ECU is built, verified and validated on time and more efficiently than by using the traditional waterfall model. To ensure that the designed system (vehicle system with its subsystems) is working well, it is essential to check whether the model respects the requirements. The model created with Matlab and Simulink is dynamic, a prototype. Within the Matlab environment, the physical process is transformed into a mathematical process, while Simulink makes it a model, which works in a certain way.

For a given ECU, if this package (the combination of Agile, Scrum, DevOps and MBD) gives a positive result, that ECU is working correctly. Moreover, this package enables to have integrations in the created model and therefore to improve its performance.

The goal is to evaluate well the single ECU placed inside the vehicular system. If the single control unit is verified and validated, the next step is the integration of two or more control units provided by different suppliers. Successful integration is a guarantee for durability, increased improvement, reliability and robustness of the final product. For the company, this package increases productivity, the way of thinking and working, provides on-time delivery, which means it makes the company attractive and trustworthy for its clients and customers.

Nowadays, the Agile methodology is the choice for many companies, not only the software developing or engineering company, but also other companies from different fields. The main reason is that it overcomes the challenges of the heavyweight methods, like the waterfall lifecycle model. It brings a new way of thinking and working, proposes flexibility. Moreover, the agile methods promote knowledge sharing and the companies obtain regular and faster feedback.

The 14th annual State of Agile report [40] collects the data from the survey conducted on more than 1000 companies from different continents, different industries and of different sizes. 95% of them adopted the Agile method, in one or more areas. The benefits are illustrated in Figure 26 [40].

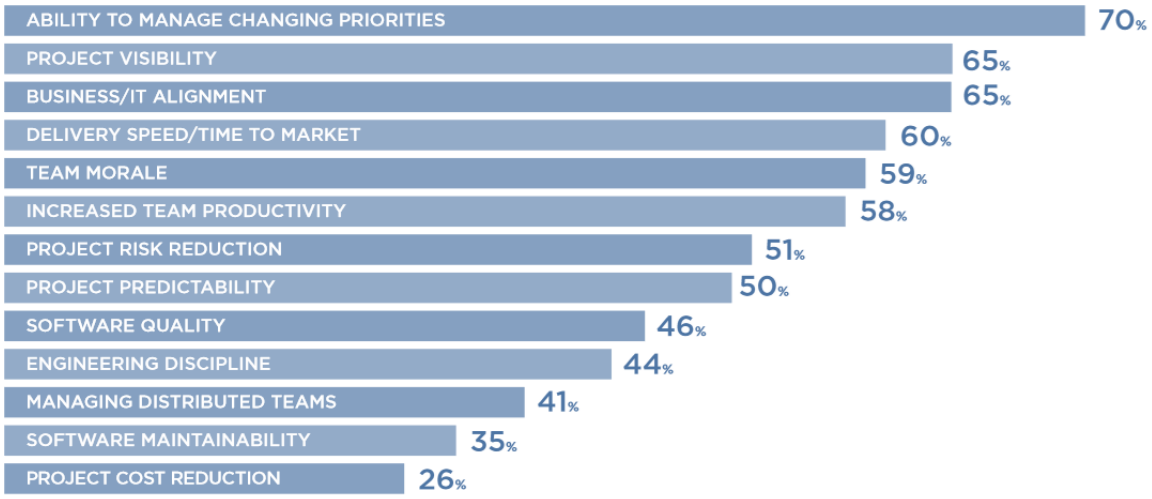

Figure 26: Benefits of adopting Agile

From these companies, 58% adopted the Scrum framework within their agile method, while 9% used multiple frameworks and practices. The combination of Agile + Scrum leads to customer/user satisfaction, on-time delivery, quality, productivity.

The report collects data also of the percentage of the companies which apply DevOps methodology. Of them, 55% already have a DevOps initiative in their organisation, while 21% are planning one. With the adoption of DevOps,

success is measured in accelerated delivery speed, improved quality, reduced risk, increased customer satisfaction, etc.

Many well-known global companies adopted one or more lightweight methods. For example, Boeing and Siemens are using Model-based development to share 3D design data, with a final goal to improve productivity and reduce the time to market [41]. Some of the companies that use DevOps and rely on communication, collaboration and innovation are Amazon, Netflix, NASA, Hertz and others [42]. Some companies transformed when they introduced Agile + Scrum methods, like Apple, Philips, Google, Spotify, IBM, Australia and New Zealand banking group, etc [43][44].

The adoption of a new method is a big step for one company regardless of its size. Before the introduction of the new approaches, the company has to overcome its challenges, to be ready for change and improvement. It is clear that there is not an ideal method or tool, which will obtain perfect results in every situation. Nevertheless, from the profound analysis that has been conducted, this proposal is likely to be the most efficient and to have the highest percentage of productivity and efficiency in specific project conditions.

Therefore, to improve the reliability of control units in the IVECO vehicles, and to fulfil the CNH Industrial requirements, that is the implementation of a unit, then integration of more units inside a subsystem, and after that, the integration of more subsystem to make the complete system, in this thesis is proposed the package Agile + Scrum + DevOps + MBD. It is expected that this new method will enhance the quality and reliability of IVECO vehicles.

Chapter 8 – Conclusion

This chapter marks the conclusion of the present thesis, developed in collaboration with CNH Industrial. The requirement from the company is to provide a suitable methodology to be implemented within the software used in the IVECO vehicles, to make it more reliable.

At the beginning of this thesis work, CNH Industrial gave directions for a method to adopt to make their vehicles reliable. For that purpose, three questions had to be answered:

- How to verify and validate the implementation of a single ECU?
- How to verify and validate the integration of more ECUs in a subsystem?
- How to verify and validate the integration of more subsystems?

Before reaching the last chapter where a complete method has been proposed, the thesis started with the description and differentiation of terms such as errors, fault and failure. Larger the presence of each of them, the reliability of the vehicular system reduces. The idea is to isolate them, especially the sporadic faults, and make them controllable. In the following chapters, the most widespread software development lifecycle models have been characterized, and then an overview of the verification and validation process has been provided. The next chapters were dedicated to Agile methodology and Scrum framework, then DevOps and lastly the Model-based development. The details provided in each chapter are important to perform a deeper analysis of the current methods. That analysis is important to approach the suggested package of methodologies and to better understand its benefits.

The package consists of Agile methodology, Scrum framework, DevOps method and Model-based development. The base of this package is the Agile methodology, a reliable method suggested by the software engineers. The other three are applied within the Agile methodology, a framework, a communication

method and a development process. This combination responds to the previously stated question and is supposed to fulfil the reliability requirements.

The electronic control unit is already a robust component. By using only one methodology for verification and validation, the reliability increases significantly. But by using the combination of these four, especially in the early stages of design and development, the robustness increases, the durability is guaranteed; therefore the overall reliability of the vehicular system is expected to increment.

The adoption of this method is not a simple step for one company; it may require partial, or even complete, reorganization in the company. The method may seem risky for companies which have adopted stable, well-known, traditional method, but the proposed methodology is rather innovative and continuously improving. From one side it can be considered as a costly method and it may increase the human and material resources, thus augmenting the overall cost. But the same method improves the trade-off between the time and the cost. Moreover, by merging the advantages of the existing methods into one, the company ensures its costumers that they use secure, safe and reliable products.

The proposed combination of methods is such as it can find its application in many other fields, not only the software and automotive engineering one.

Bibliography

[1] CNH Industrial, Website - https://www.cnhindustrial.com/en-us/know_us/who_we_are/Pages/default.aspx

[2] CNH Industrial, Wikipedia Website - https://en.wikipedia.org/wiki/CNH_Industrial

[3] IVECO, Wikipedia Website - https://it.wikipedia.org/wiki/Iveco

[4] Software Engineering, Wikipedia Website - https://en.wikipedia.org/wiki/Software_engineering

[5] IEEE Standard 610.12 (1990) *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronics Engineers

[6] Iazeolla Giuseppe (2013) *Affidabilità e Sicurezza del Software*, Franco Angeli

[7] Software Engineering, GeeksforGeeks Website - https://www.geeksforgeeks.org/software-engineering-differences-between-defect-bug-and-failure/

[8] Ahmad W., Perinpanayagam S., Jennions I., Khan S. (2014*) Study on Intermittent Faults and Electrical Continuity*, Procedia CIPR, pp. 71-75

[9] Bakhshi R., Kunche S., Pecht M. (2014) *Intermittent Failures in Hardware and Software*, Journal of Electronic Packaging, Vol.136, pp. 011014-1–011014-5

[10] Systems development lifecycle, Wikipedia Website - https://en.wikipedia.org/wiki/Systems_development_life_cycle

[11] Software development lifecycle, Techopedia Website - https://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc

[12] SDLC Tutorial, Guru99 Website - https://www.guru99.com/software-development-life-cycle-tutorial.html

[13] System development lifecycle Tutorialspoint Website - https://www.tutorialspoint.com/system_analysis_and_design/system_analysis_and_design_development_life_cycle.htm

[14] Dalbard Maurizio (2017) *Verification and Validation Techniques for the Model-Based: Gaining confidence in your design*, The Mathworks, Inc

[15] Sommerville Ian (2001) *Software Engineering*, 6th edition, Pearson, 2001

[16] Bruno Giorgio (2009). *Slides del Corso di Ingegneria del software 1*, Politecnico di Torino

[17] V-model Wikipedia Website - https://en.wikipedia.org/wiki/V-Model_(software_development)

[18] Difference between Verification and Validation, GeeksforGeeks Website - https://www.geeksforgeeks.org/differences-between-verification-and-validation/

[19] Software Inspection, Itpub Website - http://blog.itpub.net/11379785/viewspace-675909/

[20] Aldrich Jonathan (2007) *Slides on Software Inspection*, Carnegie Mellon University

[21] IEEE Standard 829 (1983) *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronics Engineers

[22] What is Software Testing and Ways of Software Testing, Toolsqa Website - https://www.toolsqa.com/software-testing/software-testing/

[23] Comparison among Black Box Testing, Gray Box Testing and White Box Testing, Software Testing Genius Website - https://www.softwaretestinggenius.com/comparison-among-black-box-testing-and-gray-box-testing-and-white-box-testing/

[24] Sandwich Testing, GeeksforGeeks Website - https://www.geeksforgeeks.org/sandwich-testing-software-testing/

[25] I principi del metodo AGILE e SCRUM e i vantaggi di applicarli all'organizzazione aziendale, Hubstrat Website - https://hubstrat.it/metodo-agile-scrum-vantaggi-azienda/

[26] Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2017) *Agile software development methods: Review and analysis* arXiv preprint arXiv:1709.08439

[27] AM Throughout the XP Lifecycle, Agile Modelling Website - http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm

[28] The Scrum Guide, Scrumguides - http://www.scrumguides.org/scrum-guide.html#definition

[29] Manifesto for Agile Software Development, Website - https://agilemanifesto.org/

[30] What is Scrum?, Scrum Website - https://www.scrum.org/resources/what-is-scrum

[31] DevOps: cos'è?, QRP Website - https://www.qrpinternational.it/blog/glossario/devops-cose/

[32] DevOps Italia: Metodologia e Filosofia Agile, QRP Website - https://www.qrpinternational.it/corsi/metodologia-devops/

[33] Lombardi C., Destri G. (2016) *I processi di sviluppo software: L'evoluzione Agile e il DevOps*, Università degli Studi di Parma

[34] Cos'è DevOps?, Miriade Website - https://www.miriade.it/cose-devops-veneto/

[35] DevOps: La rivoluzione del mondo IT, Miriade Website - https://www.miriade.it/specialisti-devops-veneto/

[36] DevOps Security Tools, Netsparker Website - https://www.netsparker.com/devops-security-tools/

[37] Kale Mangesh, Shendage Anupama, *Model-based Design for Embedded Software,* einfochips Website - https://www.einfochips.com/wp-content/uploads/resources/model-based-design-whitepaper.pdf

[38] Why is Model-Based Design Important in Embedded Systems?, einfochips Website - https://www.einfochips.com/blog/why-is-model-based-design-important-in-embedded-systems/

[39] Soltani A., Assadian F. (2016) A Hardware-in-the-Loop Facility for Integrated Vehicle Dynamics Control System Design and Validation, IFAC – PapersOnLine, vol 49, Issue 21, pp. 32-38

[40] 14th annual State of Agile report (2020), State of Agile Website - https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494

[41] Move to MBD: Can we see results?, Lifecycle Insights Website - https://www.lifecycleinsights.com/move-to-mbd-can-we-see-results/

[42] 6 Companies That Are Doing DevOps Well, helpsystems Website - https://www.helpsystems.com/blog/6-companies-are-doing-devops-well

[43] How Agile Scrum Training Transformed These 5 Companies, QuickStart Website - https://www.quickstart.com/blog/how-agile-scrum-training-transformed-these-5-companies/

[44] The Giants that use Agile: who are they?, Internetdevels official blog - https://internetdevels.com/blog/agile-lets-learn-from-the-best-ones

List of figures

List of tables

Acknowledgements

This dissertation indicates the end of my Master's studies. I would like to use this occasion to thank everyone who interfered in my university life and helped me reach my final destination, a graduate Mechatronic Engineer at Politecnico di Torino.

The most immediate acknowledgements go to my supervisor, Professor Giorgio Bruno for his determinate guidance, useful suggestions and his contribution to this thesis.

I would like to express my deepest gratitude to Engineer Raimundo Marcio Pontes, my company tutor, for solving all my doubts and giving me constructive critiques, for the patience to work with me and the life lessons. My gratitude is extended to the people from the sector “Advanced and Statistical Reliability Engineering” at CNH Industrial with their manager Enrica Vaccarino.

The greatest gratitude is reserved for my parents and my brother. Their support, emotional and financial, helped me to enrol in foreign University, to overcome each difficulty on my way and made them proud of me. Whenever they were saying: “We know you can do it!” I was a step closer to success. Although at a great distance, they survived with me all ups and downs, encouraged me and provided the strength to pursue my goal. I would also like to acknowledge the support provided by the members of my extended family.

Then I would like to thank all my friends, in Macedonia and Italy, my colleagues at Politecnico di Torino and my roommates, for their ideas, experience and all the fun we had together.

Special acknowledgement is dedicated to my boyfriend, who believed in me and is my best friend and my greatest support, who has been always by my side for the toughest moments, but also the happiest ones.