FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Virtual testing for Industrial IO modules
Student:	Bc. Erich Winkler
Supervisor:	Ing. David Holas
Study program:	Informatics
Branch / specialization:	Managerial Informatics
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2024/2025

Instructions

The goal of this thesis is to develop a virtual testing framework that allows cross-compiling of the module's firmware on Windows to test the functionality of its algorithms.
1. Get familiar with the process of testing Industrial IO modules throughout the development process
2. Identify the testing process inefficiencies in resource management
3. Propose a virtual testing framework which lowers necessary resources and improve the quality of testing
4. Implement the solution and demonstrate its functionality.
5. Evaluate the solution from a project management perspective

Electronically approved by Ing. David Buchtela, Ph.D. on 28 November 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Virtual testing for Industrial IO modules

Bc. Erich Winkler

Department of Software Engineering
Supervisor: Ing. David Holas

April 12, 2024

Acknowledgements

The path to this thesis was as long and challenging as the thesis itself. I would like to express my deepest gratitude to several people who have supported me throughout this journey and who have in one way or another contributed to the successful completion of this thesis and my studies.

I am indebted to my supervisor, Ing. David Holas, for his guidance, support and valuable feedback throughout the entire process. His expertise and knowledge had a significant positive impact on the quality of this thesis and my personal growth. The experience of working with him has been a valuable lesson that I will carry with me throughout my career and life.

I would also like to thank my family for their continuous support and encouragement. Namely, my father, Ing. Erich Winkler, who has always been a role model for me and sacrificed many of his own dreams to ensure that I am provided with the best possible education and opportunities. My mother, Ing. Lucie Winklerová, who has always been able to provide me with the necessary support and encouragement when I needed it the most. And my brother, Jáchym Winkler, who was always there to understand me and cheer me up when no one else could.

Finally, I would like to thank all of my friends and everyone who contributed to this thesis and my studies in any way. I am grateful for the opportunities that have been given to me and the experiences that I have gained. Especially, the opportunity to study abroad in the United States, which has been a life-changing experience that has shaped me into a person I am today.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on April 12, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Winkler, Erich. *Virtual testing for Industrial IO modules.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024. Also available from: ⟨https://github.com/winkleri-23/Virtual-Testing-Framework⟩.

Abstrakt

Tato práce navrhuje Virtuální testovací framework, který si klade za cíl zlepšit efektivitu procesu testování a dostupnost testování během vývoje průmyslových I/O modulů, což vede ke snížení nákladů potřebných pro testování vyvíjených modulů. Tato práce podrobně analyzuje současný proces testování v jedné z předních firem v oboru a identifikuje oblasti, ve kterých lze dosáhnout zlepšení. Navržené řešení se zabývá nejvýznamnějšími možnostmi vylepšení ve vývojovém procesu, a to tím, že poskytuje prostředí pro testování, které zvyšuje frekvenci, dostupnost a přesnost testování. Práce rovněž zahrnuje demonstraci funkcionality virtuálního testovacího frameworku a vyhodnocuje jeho vliv na vývoj a dodání projektu se zvýšeným důrazem na perspektivu projektového managementu.

Klíčová slova Vývojový proces, Průmyslové IO moduly, PLC (Programovatelný logický řadič), Projektové řízení, Správa zdrojů testování, Automatizace testů, Testovací framework, Virtuální testování, Virtualizace

Abstract

This thesis proposes a Virtual Testing Framework that aims to improve the efficiency of the testing process and accessibility of testing during the development process of industrial I/O modules, which leads to reducing the necessary resources and time required for testing the developed modules. This work closely analyzes the current testing process in one of the leading companies in the field and identifies areas for improvement. The proposed solution addresses the most significant inefficiencies in the development process by providing a testing environment that increases the frequency, accessibility and precision of testing. The thesis also includes a demonstration of the Virtual Testing Framework functionality and an evaluation of the impact on the project delivery process with a special focus on a project management perspective.

Keywords Development process, Industrial IO modules, PLC (Programmable Logic Controller), Project delivery, Resource management of testing, Test automation, Testing framework, Virtual testing, Virtualization

Contents

Introduction **1**

1 Goal of the thesis **3**

2 State-of-the-Art **5**
2.1 I/O modules 5
2.1.1 Functionality of I/O modules 5
2.1.2 Types of I/O modules 6
2.2 Project delivery process 7
2.2.1 Project management life cycle 8
2.2.1.1 Project Initiation 8
2.2.1.2 Project Planning 10
2.2.1.3 Project Execution 11
2.2.1.4 Project monitoring and controlling 11
2.2.1.5 Project Closure 12
2.3 Fundamental testing techniques 12
2.3.1 Black box testing 13
2.3.2 Types of Black Box testing 14
2.3.2.1 Functional Testing 14
2.3.2.2 Non-Functional Testing 15
2.3.2.3 Regression Testing 16
2.3.2.4 Integration Testing 18
2.3.3 Advantages and drawbacks of Black Box testing 20
2.3.4 White box testing 21
2.3.5 White Box Testing Techniques 21
2.3.5.1 Statement Coverage technique 22
2.3.5.2 Branch Coverage technique 22
2.3.5.3 Path Coverage technique 22
2.3.6 White Box Testing Strategies 23
2.3.6.1 Unit Testing 23
2.3.6.2 Static Code Analysis 26
2.3.7 Advantages and Limitations of Whitebox testing 27
2.3.8 Gray box testing 28
2.4 I/O module development 30

2.4.1 Project delivery 30
2.4.2 Implementation process 32
2.4.3 Testing of I/O modules 33
2.5 Methods of virtualization 36
2.5.1 Levels of Virtualization 37
2.6 Convenient level of virtualization for this thesis 40

3 Analysis and Design 41
3.1 Analysis of the current testing process 41
3.1.1 Testing time frames and delays 41
3.1.2 Quality of testing 45
3.1.3 Inefficiencies of the testing process 46
3.2 Requirements engineering for the VT framework 48
3.2.1 Developer's requirements 48
3.2.2 Tester's requirements 49
3.2.3 Management's requirements 50
3.2.4 Outcome of the requirements engineering process 51
3.3 Virtual testing framework 52
3.3.1 Architecture 52
3.3.2 Virtual testing library 54
3.3.3 Server application 57
3.3.4 Testing client 58
3.3.5 Communication & Interfaces 59
3.4 VT Framework's Impact from a Project Management Perspective 60
3.4.1 Project planning impact and risk reduction 60
3.4.2 Unified testing strategy 62
3.4.3 Financial impact 63
3.4.4 Analysis summary 69

4 Implementation 71
4.1 Virtual testing library 71
4.1.1 Emulator 72
4.1.2 DLL interface 75
4.1.3 LLC communication layer 75
4.1.4 Sample module 76
4.2 Server application 78
4.3 Testing client 79
4.4 Logger 81

5 Demonstration of the VT Framework usage 83
5.1 Prerequisites & Installation 83
5.2 Test case preparation 83
5.3 Module parametrization 85
5.4 Diagnostics testing 85
5.5 Logging & Reporting 88

6 Evaluation of the Virtual testing framework 91
6.1 Requirements fullfillment 91
6.2 Resourse management 94
6.3 Evaluation summary 95

6.4 Suggestions for future improvements 96

Conclusion **99**

Bibliography **101**

A Acronyms **105**

B User Manual **107**
B.1 Prerequisites 107
B.2 Project preparation 107
B.3 Running the Virtual Testing system 107

C Attachments Contents **109**

List of Figures

2.1 Project initiation activities 10
2.2 Representation of Black Box Testing 14
2.3 Top Down Integration approach 19
2.4 Big bang Integration testing approach 20
2.5 Statement Coverage formula [13] 22
2.6 Branch Coverage formula [13] 22
2.7 Test phases of the I/O module development 34
2.8 Internal release - Test workflow 35
2.9 Levels of virtualization[24] 37

3.1 Process of fixing bugs discovered during testing phases 44
3.2 Scope of testing 48
3.3 Architecture of the VT framework 53
3.4 Architecture of the Virtual testing library(DLL) 57
3.5 Architecture of the testing client 59
3.6 Low-level command 60
3.7 Budget distribution 64
3.8 Automated testing in CI/CD pipeline - Feedback loop 70

4.1 Directory structure - VT library 72
4.2 Representation of an event in the event calendar 73
4.3 Implementation of the execute function 74
4.4 Communication layers 79
4.5 Example of the logger usage 82

5.1 Example of the test case for the sample module 85
5.2 Simplified example of the test case for the sample module 86
5.3 The flow of the Test_WB Test case 87
5.4 Command line argument for setting the logging level 88
5.5 Logging macro for the sample module 88
5.6 Logging output of the VT library and the server application 89
5.7 Logging macro for the sample module 89
5.8 Output of the logging macro for the sample module 89

6.1 Time needed to implement one test case 93

List of Tables

2.1 Regression testing techniques - comparison [10] 18

3.1 Time frames for testing phases 42
3.2 Potential ROI for integration testing - discovered bugs 66
3.3 Potential ROI for system testing - discovered bugs 68

Introduction

Developing any software or hardware product is a complex process that requires a lot of effort and resources. The development of I/O modules is no exception in this regard, as they often control critical systems and control heavy machinery. For this reason, many standards and regulations must be followed during the development and testing to ensure the safety and reliability of the final product.

Over the years, as the complexity of the requirements and the products themselves have increased, it has become costly and time-consuming to ensure that the products meet all of them. Therefore, all companies are looking for ways to reduce the cost of testing while maintaining or even improving the quality of the products.

It is clear that the traditional testing methods are not sufficiently scalable and efficient. They often require a lot of manual work and are not easily adaptable. The technology has advanced significantly in recent years, and therefore, it is possible to automate the testing process to a large extent by creating a way to test the behavior of the I/O modules without the need for physical hardware and with minimal human effort.

However, the solution to the costly testing on the physical hardware does not lie in replacing the physical testing entirely but rather in complementing it with other methods. The current testing methods are still necessary and required by the certification authorities that allow the products to be sold on the market. However, by reducing the number of issues found late in the testing process on the physical hardware, the cost of fixing them is also reduced.

The objective of this thesis is to create and put into practice a Virtual Testing Framework that can effectively address the issues described above. This framework would significantly enhance the quality of testing for one of the largest corporate companies in the European Union, where I am employed. Unfortunately, the company's name cannot be disclosed in this thesis due to internal policies, and will therefore be referred to as "The Company".

The Virtual Testing Framework is designed to be a flexible and scalable solution that improves the accessibility of testing, allows the creation of a wide range of test scenarios, and provides a way to automatically test the behavior of the modules after each change in the code. The requirements for this project are based on the needs of the company that have been identified by a thorough analysis of the current project delivery process.

With all stated above, the Virtual Testing Framework is expected to be

a valuable tool for developers and testers in the Company. It will help them to deliver high-quality products faster and more efficiently. In addition, it provides a way to improve the predictability of the development process which is a crucial aspect for the project managers and stakeholders in the Company.

Chapter 1

Goal of the thesis

This thesis aims to create a Virtual Testing Framework that can effectively increase the efficiency of the project delivery process of I/O modules in the Company. In order to achieve this goal, the following steps need to be followed.

Firstly, get familiar with the current testing throughout the development process in the Company. This includes understanding the testing requirements, current methods, and the overall workflow of the testing process in the Company.

Secondly, analyze the current testing process and identify the areas where the testing can be improved. This step of identifying the bottlenecks is crucial for deriving the requirements for the Virtual Testing Framework.

Thirdly, based on the requirements identified in the previous step, propose a solution that addresses the identified inefficiencies and therefore lowers the required resources for testing while improving its quality.

Lastly, implement the proposed solution, demonstrate its capabilities, and evaluate its impact from the project management perspective. That includes the time saved on testing, the quality of the testing, and the overall improvement in the project delivery process.

Chapter 2

State-of-the-Art

This chapter describes the term I/O module and its development process as well as the project management lifecycle. It primarily focuses on the current testing processes and points out the importance of all testing phases. Furthermore, it introduces the reader to the current methods of virtualization that can be used for testing purposes and determines which could be beneficial for this thesis.

2.1 I/O modules

This section provides an overview of the I/O modules. It introduces the reader to the concept of I/O modules, their types, and their typical functions in an industrial automation system. This knowledge is essential for understanding the whole concept of the Virtual Testing Framework, which is the main topic of this thesis.

Introduction to I/O modules
The I/O modules, otherwise known as Input/Output modules, are devices that manage communication between PLC (Programmable Logic Controllers) and network, including data exchange, power load management, and machine control functions.

That allows the system integrators to interconnect various devices, providing enhanced control over the industrial network. The I/O modules have become an essential part of industrial automation systems, especially in instances where there exists large machinery, devices, or any systems that are unable to communicate with a desired industrial protocol on their own. Examples of such devices include sensors, actuators, and monitors. Besides that, I/O modules typically work as accessory devices such as PLCs. [1]

2.1.1 Functionality of I/O modules

The I/O modules have a variety of functions that are essential within the industrial environment. They enable to incorporate all manufacturing devices into a single network, enabling greater control of the system as well as increased visibility.

[1] The core functions of I/O modules that make them valuable in industrial automation systems are:

- **Error Detection:** I/O modules can detect errors, typically using parity bit methods.
- **Processor Communication:**
  - **Command Decoding:** Receives and decodes commands from the processor.
  - **Data Exchange:** Transfers data between peripherals, processors, and main memory.
  - **Status Reporting:** Communicates peripheral status to the processor.
  - **Address Decoding:** Organizes peripherals by managing unique addresses.
  - **Data Buffering:** Manages data transfer speed between processor and peripherals, compensating for latency.
- **Device Communication:** Facilitates communication between connected peripheral devices.
- **Control and Timing:** Manages data transactions between internal system and peripherals.

2.1.2 Types of I/O modules

I/O modules can be categorized into several types, depending on their functionality and the type of data they handle. The most common types of I/O modules are [2]:

- **Digital modules:** These modules handle digital data, such as on/off signals, and are used to control devices like motors, lights, and switches.
- **Analog modules:** Analog modules handle continuous data, such as temperature, pressure, and flow rate, and are used to control devices that require variable input signals.
- **Communication processors:** These modules manage communication between devices on a network.
- **Fail-safe I/O modules:** Fail-safe I/O modules are designed to ensure that critical systems continue to operate in the event of a failure, such as a power outage or network disruption.

Commonly, each module offers its digital and analog variants to provide a wide range of options for system integrators. To understand the differences between these modules, it is essential to understand the difference between the digital and analog signals. The analog signal is a continuous signal that varies over time and can take any value within a specified range. In contrast, the digital signal is a discrete signal that can take only two values, typically represented as 0 and 1. Even though the majority of modern devices are digital,

analog signals are still used in many industrial applications. In addition to that, the analog signal is easily convertable to a digital signal which allows to ensure compatibility between different devices.

2.2 Project delivery process

Even though the VT Framework influences mainly the development itself, it is essential to understand the fundamentals of project management to recognize the benefits of the framework for project planning. For this reason, this section introduces the reader to project management and describes all the essential phases of the project management life cycle. Following the introduction to project management theory, the focus will shift to the specific details of the project delivery process of the I/O module projects and the I/O module development in the Company.

To understand what the project delivery process is, we need to first define the term project itself. We need to clarify what are the important aspects of this term and what it means from the project management point of view.

The term **project** is defined as follows: "*A project is a unique endeavor to produce a set of deliverables within a clearly specified time, cost and quality constraints.*" [3] According to this definition, a project is defined by the following aspects: [3]

- **Unique:** A project is unique and does not involve repetitive processes. That makes it different from standard business operational activities as they often consist of identical processes.
- **Timescale:** Every project should have a clearly defined beginning and end date for its delivery.
- **Budget:** Projects are given a budget to create deliverables that meet customer needs.
- **Resources:** A specific amount of labor, equipment and materials is allocated to the project at its beginning.
- **Risk:** Projects involve a level of uncertainty and consequently carry business risk.
- **Beneficial change:** The purpose of any project is to achieve a beneficial business change.

Now that we have defined the term *project* let us explore what project management is all about. This term can be defined in various ways, and we will take a look at a couple of the best examples. Asana, Inc., a company that created one of the most significant work management platforms, defines this term as follows: In the context of this thesis, we define the term as follows: "*IT project management is the process of managing, planning, and developing information technology projects. IT projects exist within a variety of industries, including software development, information security, information systems, communications, hardware, network, databases, and mobile apps*" [4] This definition gives us a good overview of what we should imagine under project management in practice. However, the usage of this definition is limited, and it does not allow

us to focus on project management theory. For this reason, another definition of this term comes to light.

Jason Westland, in his book, defines project management in the following way: "*Project Management is the skills, tools and management processes required to undertake a project successfully.*" According to this simple definition, project management incorporates: [3]

- **A set of skills:** Expertise, specific skills, and practical experience are essential for mitigating project risks and increasing the chances of its successful completion.
- **A suite of tools:** To enhance the probability of success, project managers use various types of tools such as planning software, modeling software, audit checklists and review forms.
- **A series of processes:** To effectively oversee and regulate project parameters such as time, cost, quality, and scope, various processes and techniques must be employed. Examples include time management, cost management, quality management, change management, risk management and issue management.

Given these definitions, we can shift the focus of this section to the project life cycle.

2.2.1 Project management life cycle

There are many different types of projects, however, they all share a common characteristic. They all follow the same cycle known as the project management life cycle. It consists of five phases otherwise called process groups, which are universal to all projects. However, the specific phases within a project are unique to each project, shaping its unique life cycle. [5]
The five phases of project management life cycle are:

1. **Project Initiation**
2. **Project Planning**
3. **Project Execution**
4. **Project Monitoring and Controlling**
5. **Project Closing**

2.2.1.1 Project Initiation

In this phase, the first step involves identifying a business problem or opportunity. Once identified, a solution is defined, and a project is established. A project team is then appointed to develop and deliver the solution to the customer. In order to achieve that, multiple activities displayed in Figure 2.1 need to be undertaken during this phase in the correct order.

Develop a business case
This activity starts with identifying a business problem or opportunity. In other words, an entity in the firm addresses an opportunity and comes up with the idea of starting a project. Then, a comprehensive business case needs to be created to define the problem or opportunity clearly and in detail and identify a preferred solution for implementation. [3]

[3]The business case typically includes the following attributes:

- A thorough explanation of the problem or opportunity.
- A list of the various available solutions.
- An analysis of the business benefits, costs, risks, and issues.
- A description of the favored solution.
- A plan outlining the implementation process.

Undertaking a feasibility study
Once the business case is developed, a follow-up activity takes place. It is mandatory to finish the business prior to initiating this phase. A well-executed feasibility study guides a project by providing decision-makers with a comprehensive understanding of potential advantages, drawbacks, obstacles, and limitations that may impact its outcome. The primary purpose of a feasibility study is to assess whether the project is not only technically, financially, legally, and market-wise viable but also beneficial. The findings of your project feasibility study are typically compiled in a *feasibility report*. [6]

Establish the terms of reference
Once the business case and feasibility study have been approved, a new project is formed and another activity can be initiated. The goal of this phase is to create a *terms of reference*(ToR). This crucial document outlines the vision, objectives, scope, and deliverables of the new project. In other words, should clearly define the anticipated outcomes and deadlines for the project sponsor, as well as outline the expectations for key stakeholders and task force members regarding the deliverables and the approach to project execution. [3]

Appoint the project team
The objectives of the projects are now clearly defined. Therefore, it is clear what sets of skills are going to be necessary to deliver the project. Although a project manager may be appointed at any stage during the life of the project, it is a good practice to do it prior to recruiting the project team. The project manager usually creates a detailed job description for each role in the project team and recruits people into each role based on their relevant skills. [3]

Set up a project office
The project office serves as the physical space where the team is situated. While a central project office is common, the option of establishing a virtual project office exists, allowing project team members to be geographically dispersed across the globe.

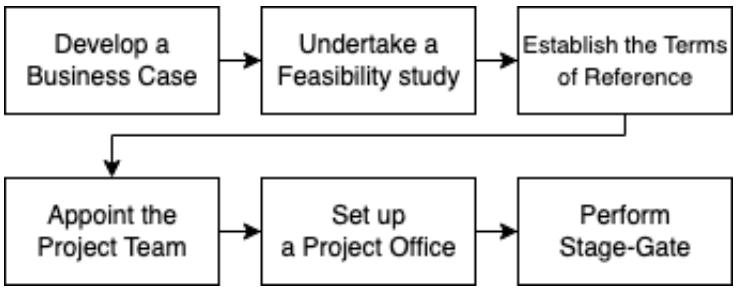

Figure 2.1: Project initiation activities [3]

2.2.1.2 Project Planning

By now, the project costs and benefits have been documented, outlined its objectives and scope, and the team has been appointed. At this point, it is time to undertake detailed planning to ensure that the activities performed during the execution phase of the project are properly sequenced, resourced, executed, and controlled. Since the primary subject of this thesis does not have a direct impact on the project planning process, we will not go into exhaustive details about each activity in this project phase.

[3]Following activities need to be performed during the planning phase:

- **Create a project plan:** A Work Breakdown Structure (WBS) is identified - a hierarchical set of phases, activities and tasks to be performed to complete the project. Once the WBS is agreed upon, the detailed project plan is formed and used as the key tool by the project manager to assess the progress of the project.

- **Create a resource plan:** Allocation of the required types and quantities of resources, specifying human roles and skill sets, detailing equipment specifications, and listing material requirements, with scheduled reviews by the project manager at each project stage.

- **Create a financial plan:** The financial plan, crucial for project management, establishes the budget by identifying the total cost of labor, equipment, and materials for each project phase.

- **Create a quality plan:** A quality plan ensures meeting customer expectations by defining project quality, setting clear deliverable targets, outlining a plan for assurance, and identifying techniques used to control the actual quality level of each deliverable.

- **Create a risk plan:** The risk plan is a document that identifies and describes all foreseeable project risks. Besides that, it also identifies the actions required to prevent each risk from occurring. A clear risk plan is crucial for any project as it is necessary to be aware of all the crucial risks and know how to mitigate them prior to entering the execution phase.

- **Create an acceptance plan:** An acceptance plan is formulated to ensure successful project delivery, specifying completion criteria for each deliverable and establishing a schedule for acceptance reviews. This enables the customer to assess and formally approve deliverables, ensuring they meet or exceed initial requirements.

- **Create a communication plan:** Before the execution phase, a communications plan is developed to outline how stakeholders will be informed about the project's progress. It specifies the types, methods, and frequency of information distribution, along with assigning responsibilities within the project team for communication tasks.

- **Create a procurement plan:** The procurement plan offers a comprehensive overview of the products (goods and services) to be obtained from external suppliers. It includes the justification for acquiring each product externally as opposed to from within the business, and the schedule for product delivery.

- **Contact the suppliers:** Appointing external suppliers is a flexible aspect of the project timeline; however, the standard procedure involves their appointment after formalizing project plans but before the execution phase. At this point, a formal tender process is undertaken to identify a short list of capable suppliers and select a preferred supplier to initiate contractual discussions with.

2.2.1.3 Project Execution

Once the Project plan is created, the most important phase of all takes place. The execution phase is typically the longest phase of the project and determines whether your project will succeed or not.

In this phase, the deliverables of the project are created by executing the project plan. The activities undertaken to fulfill project requirements will vary depending on the project type. Tasks or activities can follow a *waterfall* approach, where each activity is completed in order until the final deliverable is achieved, or an *iterative* approach, where the product is developed and tested in smaller segments of work called iterations. Regardless of the method the project manager chooses, it is crucial to carefully monitor and control processes to ensure the quality of the final deliverable. [3]

2.2.1.4 Project monitoring and controlling

In project management theory, the monitoring and controlling phase is often considered part of the execution phase. The reason is that this phase measures the project's performance, tracks progress, and, therefore, occurs simultaneously with execution. The main goal is to continuously verify alignment with the project management plan, focusing mainly on financial aspects and timelines.

It is the responsibility of the project manager to make necessary adjustments related to resource allocation and ensure that everything stays on track, meeting the customer's requirements.

[3]To achieve this goal, the following management processes need to be undertaken:

- **Cost Management:** Process of identifying, approving, and paying project costs. Expense forms are completed for labor, equipment, and materials.
- **Quality Management:** Ensures final deliverables conform to customer requirements. Quality is assured and controlled through quality assurance and quality control techniques.
- **Change Management:** Process of formally requesting, evaluating, and approving changes to project scope, deliverables, timescales, or resources. The project manager manages change by understanding business and system drivers, identifying costs and benefits, and formulating a structured plan.
- **Risk Management:** Process of formally identifying, quantifying, and managing risks to the project.
- **Issue Management:** Method of formally managing issues affecting the project's ability to produce required deliverables. Issue forms are completed and logged in an issue register. The project manager evaluates each issue and takes action to resolve it.
- **Acceptance Management:** Process of gaining customer acceptance for project deliverables produced by the project.

2.2.1.5 Project Closure

The last phase of the PLM (Project Management Lifecycle) consists of two equally important activities: **Project closure** and **Project Completion**. The project closure includes delivering the final deliverables to the satisfaction of the stakeholders, transferring project documentation, terminating supplier contracts, releasing project recourses and informing all stakeholders and interested parties of the project's closure.

Once the project closure is done, it is time to perform project completion. This final step includes an independent assessment of the project's success based on performance against defined objectives conformed to the management process outlined in the planning phase. Results of the review, as well as a list of key achievements and lessons learned, are typically documented within a post-implementation review and they are required to be approved by the customer. [3]

2.3 Fundamental testing techniques

In the realm of software development, testing plays a crucial role in ensuring the reliability and quality of the products. There are three fundamental approaches to software testing, namely Black box testing, Whitebox testing and their combination called Graybox testing. This chapter aims to provide a comprehensive understanding of these testing techniques, the scope of the

testing they cover, and the main differences between them. This section covers the fundamentals of the testing techniques without specific details about I/O module development. It introduces the reader general overview for an easier understanding of the specific details of the usage of the Virtual Testing Framework that will be described later in this thesis.

2.3.1 Black box testing

This software testing technique focuses on evaluating the functionality of a software application without the knowledge of its internal code structure. In other words, the tested program or system is treated as a "Black box", and the focus lies on examining the available input for an application and the expected outputs corresponding to each input value.

Testers create tests based on software requirements and specifications, typically operating with no insights into the internal workings of the software under examination. Hence, this testing method is often referred to as behavior testing. In other words, the testers do not need to know anything about the internal code implementation of the tested item. The main goal of this testing is to compare the observable behavior from the "outside" and compare it with the expected behavior for both, valid and invalid inputs, described in software specifications.[7]

[7]Black box testing techniques:

- **Equivalence partitioning:** This testing technique involves a systematic division of all input values into partitions. Both, valid and invalid partitions are included and test cases are designed and created for each partition to reveal potential errors.
- **Boundary Value Analysis:** This method focuses on boundary values or nearby boundary values of input data. Test cases are typically designed for valid and invalid inputs to ensure the tested item can maintain the expected behavior for all possible inputs.
- **Decision Table Based Testing:** This technique proves effective in handling a substantial volume of inputs and their corresponding outputs. The decision table has a completeness property, which means it contains all possible values of condition variables.
- **Error Guessing:** As the name of this technique suggests, it involves making assumptions and educated guesses about potential defects. The success of this approach heavily relies on the experience and proficiency of the testers.

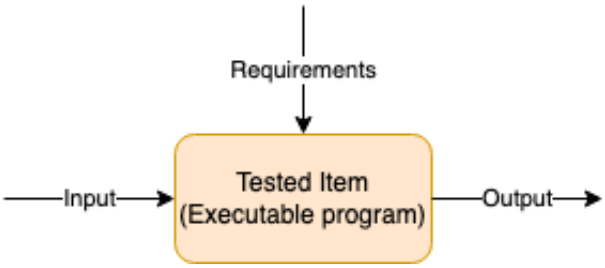

Figure 2.2: Representation of Black Box Testing [7]

2.3.2 Types of Black Box testing

Various factors such as testing objectives, techniques employed, and the desired level of detail contribute to the categorization of black box testing into different types. Functional testing, non-functional testing, regression testing, and integration testing are among the common classifications within this testing methodology.

2.3.2.1 Functional Testing

Functional testing is a crucial aspect of black box testing, focusing on validating whether the software application aligns with the *functional* requirements outlined in the design documents. It employs a variety of testing techniques and methodologies to guarantee that the system's functionality works correctly. Common types of functional testing are smoke testing, sanity testing, integration testing, system testing, and regression testing. [8]

Smoke testing is typically performed in the early stages of the software development life cycle. This testing aims to ensure that the system's critical functions work as expected. The focus lies on the major features of the tested software with minimal test cases. This testing is typically followed by **Sanity testing**, which ensures that the tested system is stable and ready for further testing.

Integration testing assesses the interactions and interoperability among modules and components within the tested system. The goal is to confirm the proper operation of the integrated system, ensuring that individual components collaborate seamlessly. Since it is essential for the development of the I/O module and will be explored in the following chapters in a convenient context.

System testing evaluates the entire system to determine its compliance with specified requirements. This comprehensive evaluation involves testing the system from end to end, verifying that all components and functionalities work together according to expectations. Similar to integration testing, this type of essential for I/O module development and therefore will be explored in the following chapters.

Regression testing ensures that any new changes do not negatively impact the existing functionality of the system. The process includes retesting functionalities that were previously tested, along with the newly implemented changes. That is because any new code may bring in new logic that conflicts

with already existing code, which usually leads to bugs and unexpected behavior.[8]

2.3.2.2 Non-Functional Testing

Nonfunctional testing is just as crucial as functional testing for ensuring the overall quality and performance of an application. While functional testing concentrates on testing the specific functionalities of the product, nonfunctional testing assesses performance and usability parameters. In other words, this type of testing verifies nonfunctional aspects of the product, such as performance, stability, and usability, rather than testing whether the product does what it is supposed to do. [8]
[9] **Types of Non-Functional testing:**

- **Performance Testing** assesses responsiveness and stability under specific workloads, providing insights into scalability, reliability, and resource usage.
- **Load Testing** ensures that a system can effectively function under specific loads, whether it involves large data quantities or a large number of users.
- **Stress Testing** assesses the reliability of the product under unexpected or rare workloads, pushing the system beyond normal operational limits to determine its robustness. The primary goal is to ensure the product remains stable and does not crash, even in conditions of insufficient computational resources.
- **Stability Testing** assesses if the product can consistently function within acceptable time frames.
- **Usability Testing** ensures that the system's intended users can perform tasks efficiently, effectively and satisfactorily.
- **Compatibility Testing** ensures that the tested product is compatible with various elements such as web browsers, hardware platforms, operating systems or any other components required in product specification documents.
- **Endurance Testing** involves testing the product with a significant load extended over a long period of time. This testing is particularly important for I/O module development as the memory limitations may lead to algorithms relying on counters that can gradually exceed over time.
- **Security Testing** is performed to ensure the product aligns with the security requirements. Six basic security concepts are usually covered by this type of testing - confidentiality, integrity, authentication, availability, authorization and nonrepudiation.

In conclusion, nonfunctional testing is essential for ensuring the product's overall quality, performance, compatibility, and user experience. It ensures that the product is convenient for the intended usage by the customer. It is worth

mentioning, that the nonfunctional requirements might differ product from to product and the test framework needs to be adapted to these requirements.

2.3.2.3 Regression Testing

Regression testing is a very specialized branch of software testing that focuses on validating that all the functionality remains after recent modifications were made. It typically includes performing previously conducted test cases to verify the continued functionality of existing features. By testing newly implemented code changes, it is ensured that no unforeseen impacts on current functionality occur. It is a good practice to debug the code and potential issues prior to performing the regression testing process. The testing is performed on a subset of relevant test cases from the existing test suite, that includes both updated and revised code.

There are several techniques for regression testing, each distinguished by the method used to test cases for your test suite. Each of the following test selection techniques has been studied for real-world usage and each is convenient in different situations. [10]

Retest All

A popular and probably most intuitive regression testing technique includes re-running all the test cases from the existing test suite. This retest-all technique aims to validate the entire tested product however it can be prohibitively expensive in terms of both time and financial resources. Besides that, it is highly impractical for individuals to perform all the tests manually due to the vast amount of testing required. Consequently, it is a common strategy to automatize as many tests as possible so they can be easily performed without any human interaction. That allows for a significant increase in the number of test cases that can be performed in a given amount of time and therefore increases the testing frequency. [10]

Regression Test Selection

Even though the automatization of the test cases allows us to run more test cases in the same time period, it can still be costly and time-consuming. For this reason, an alternative selection technique was proposed. Instead of retesting the complete test suite, only a subset of the tests is performed. That allows testers to skip costly test cases that are unlikely to discover any error. The existing test suite is typically categorized into three groups: reusable test cases, retestable test cases, and outdated test cases.

The regression test selection technique can also introduce additional test cases, thereby enhancing the testing of the program, particularly in areas not covered by existing test cases. There are three commonly known types of test selection techniques: [10]

- **Coverage techniques:** These techniques involve considering the entire test coverage criterion. They select the most appropriate test cases and exclusively perform testing on software components that have undergone alternations or updates.

- **Minimization techniques:** These approaches are similar to the previously mentioned coverage technique. However, they perform testing by specifically choosing a limited number of test cases that are sufficiently relevant for the intended purpose.
- **Safe techniques:** These techniques do not prioritize the inclusion of all software requirements. Instead, it focuses on incorporating all possible software test cases that are expected to generate different results with a small change in the original version of the program.

Test Case Minimization

This technique focuses on reducing the number of test cases required for the testing process while still maintaining the capacity to detect faults and bugs. That is achieved by identifying and deleting redundant test cases based on specific test adequacy criteria. These approaches create a representative set from the original test suite, ensuring it fulfills all defined requirements with the minimum number of test cases. Removing redundancy from the test suite decreases overall testing effort and therefore lowers the cost of the regression testing. [10]

Test Case Prioritization

This regression testing technique prioritizes test cases, enhancing the efficiency of defect identification within the test suite and subsequently improving dependability. For this approach, existing test cases are typically categorized based on various factors, including coverage-based criteria, financial cost, previous experience, and customer requirements. [10]

To sum it all up, let us take a look at the comparison of various regression testing approaches summarized in the following table 2.1

Test Strategy	Regression Test Approach		
	Selection	**Minimization**	**Prioritization**
Method	Identify the test cases that are relevant to a particular set of recent changes.	Remove test cases that are redundant and unnecessary.	Sort the test cases based on specific criteria so that the more effective cases concerning those criteria are executed first.
Benefits	Test cases that recognize modifications are more inclined to be chosen.	It has been observed to be effective in reducing the number of test cases.	Newly generated test cases are consistently reviewed in the test case permutation.
Limitations	There is a high possibility that the new test cases could be overlooked because of temporary selections that are modification-aware.	Test cases lack awareness of modifications.	The chosen test suite is both extensive and time consuming.

Table 2.1: Regression testing techniques - comparison [10]

2.3.2.4 Integration Testing

Integration testing is a crucial part of the software development process, focusing on the interactions and interoperability among modules and components within the tested system. It can be considered both, White box and Black box testing, as it involves testing the internal structure of the system as well as the external behavior. The goal is to examine that the individual components collaborate together as expected when they are combined to form a part of a larger system. It is crucial to thoroughly test the individual components prior to the integration testing, as it is essential for all the components to be stable and reliable. The testing process of the individual components is typically performed using the White box testing technique called Unit testing, which will be described later in this thesis.[11]

[11] **Integration testing is typically performed in 4 different ways:**

- **Top-Down Integration:** This method begins with the testing of higher-level units first, followed by the testing of lower-level units. This approach is typically used when the lower-level modules are not available. The missing unit is given to a 'stub', which temporarily replaces the missing module by providing the desired interfacing data flow.
- **Bottom-Up Integration:** This method begins with the testing of lower level units first. Once all the units at the lower level are tested, the testing continues to the higher level units. In case the higher level modules are not available, they are replaced by a 'driver' similar to the 'stub' in the top-down approach. [12]
- **Big Bang Integration:** This method is quite different from the previous two. Instead of integrating the system module by module, all the units are integrated simultaneously and tested as a whole.
- **Sandwich Integration:** This method is a combination of previously mentioned top-down and bottom-up approaches. It uses both 'stubs' and 'drivers' to replace the missing modules as the testing starts somewhere in the middle of the module hierarchy.

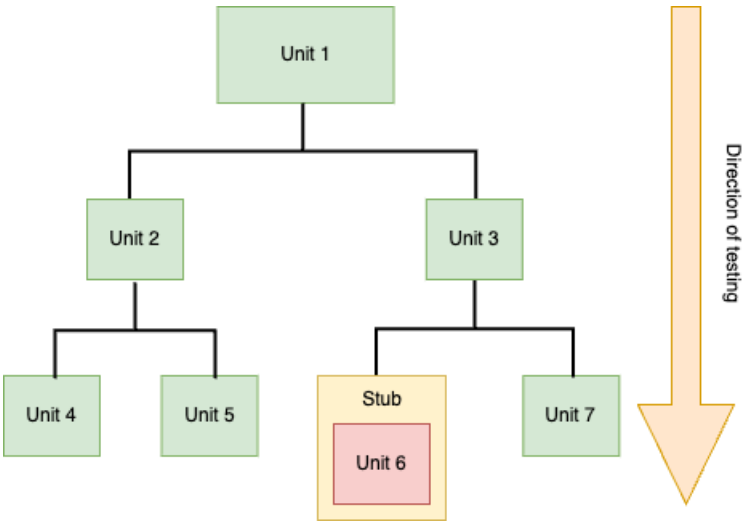

Figure 2.3: Top Down Integration approach [11]

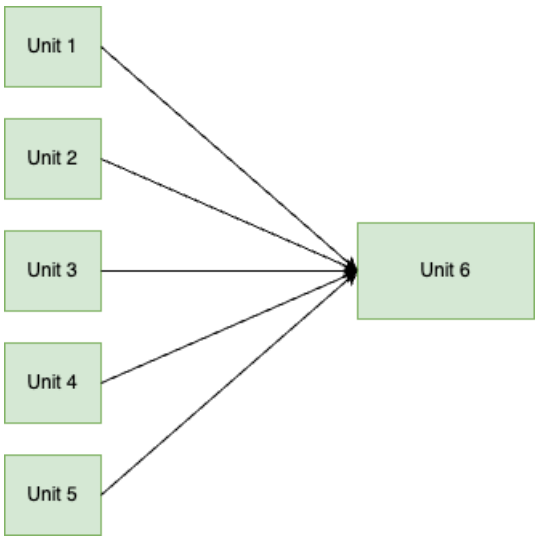

Figure 2.4: Big bang Integration testing approach [11]

To sum it all up, the goal of the integration testing is to expose faults in the interaction between integrated units. It usually follows the unit testing process which will be discussed in the following chapters. It is important to understand that integration testing includes both, Black Box and White Box testing, as sometimes it is necessary to test the internal structure of the system as well as the external behavior.

2.3.3 Advantages and drawbacks of Black Box testing

Properly designed and executed black-box testing brings numerous benefits that we will cover in this section. However, it is essential to use them in suitable situations and be aware of their limitations.

Black box testing allows us to perform testing according to the customer's requirements. Therefore, it gives the developers and the customer clear information if the developed product behaves as expected. This is closely tied to another advantage, which is very minimal requirements on the tester's implementation knowledge. Tests are typically carried out by a third party and specialized testers who do not need to possess in-depth knowledge of the implementation itself. This allows companies to have specialized developers and testers, leading to better efficiency. The following list summarizes the most significant advantages of this testing technique. [7]

[7]Advantages of Black box testing:

1. **Based on customer's requirements:** Testing is conducted based on the requirements outlined from the customer's perspective.
2. **Independent Evaluation:** Testing is usually performed by a third party (independent tester team) to avoid developer bias.

3. **Implementation knowledge is not required:** Testers do not need to possess any implementation knowledge about the tested product to perform Black box testing.

4. **Scalability:** Testing proves efficient when applied to larger systems, and test cases can be designed as soon as specifications are finalized.

All of these benefits make this technique a crucial step in the software development lifecycle. However, it is important to consider some limitations. As mentioned earlier in this chapter, the process of designing and creating new test cases heavily depends on well-defined requirements. If the requirements are poorly described, it becomes challenging to create and execute tests accurately. Even with clear requirements, creating test cases that cover all possible scenarios is extremely time-consuming. Additionally, despite having a properly designed set of test cases, it may not uncover all defects in the system, as there is no way to test the functionality of the internal workings of the tested item. [7]

2.3.4 White box testing

White Box Testing is a software testing method that focuses on the internal logic, code structure, and control flow of the tested item. As all the alternative names such as clear box testing, open box testing, and transparent box testing suggest, the main idea is to test the software with a good understanding of its internal structure and with the ability to see what is happening inside the application.

This testing method typically examines all paths of the source code. That requires strong programming skills and an understanding of the application's architecture. The test cases are derived from an analysis of the system's internal structure, focusing on aspects like code coverage, branch coverage, path coverage, and condition coverage. Key characteristics of White Box Testing include: [13]

1. The tester possesses a complete understanding of the software's internal operations.

2. It allows for effective testing of data domains and internal boundaries.

3. It is particularly suitable for Algorithm testing.

4. It is primarily conducted by testers and developers.

5. It offers a high level of granularity.

2.3.5 White Box Testing Techniques

White Box Testing offers a variety of techniques that ensure that as many as possible paths and conditions within the software are tested. This section introduces the most common of those techniques and explains their main principles.

2.3.5.1 Statement Coverage technique

Statement coverage testing technique plays a crucial role in ensuring the accuracy and precision of software testing. To understand this method, it is important to explain the term 'Statemen coverage' first, which is a metric used to measure the number of executed statements in the source code. The formula for the statement coverage is shown in the following equation:

$$\text{Statement Coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100 \tag{2.1}$$

Figure 2.5: Statement Coverage formula [13]

The main goal of this technique is to execute all the executable statements in the source code at least once. Even though it is a robust method, it may not encompass all potential bugs and issues. Particularly those that are related to specific conditions or decision outcomes. That is why it is important to combine this technique with other testing methods such as branch coverage and path coverage. [13]

2.3.5.2 Branch Coverage technique

The branch coverage testing method is also commonly known as decision coverage or edge coverage. As the name suggests, the main goal of this technique is to test all the possible branches in the source code. That means every branch needs to take the true and false paths at least once. The main advantage of this approach is that it can detect any potential issues related to the decision-making process in the source code. The formula is similar to the statement coverage formula as you can see in the following equation:

$$\text{Branch Coverage} = \frac{\text{Number of executed branches}}{\text{Total number of branches}} \times 100 \tag{2.2}$$

Figure 2.6: Branch Coverage formula [13]

2.3.5.3 Path Coverage technique

As the name suggests, the tests are designed to ensure that every possible path is executed at least once. The aim is to test each unique executable path as thoroughly as possible to maximize the coverage of each test case. This method allows both test cases and their outcomes to be mathematically analyzed, leading to a precise measurement. For instance, if a method has N decisions, it could potentially have 2^N paths. However, it is often impractical to perform the whole path coverage testing, and it is necessary to identify useful subsets of paths to be tested. The following strategies have been developed for this purpose: [13]

- **Basis Path Testing** ensures that each statement in the program is executed at least once. Creating and executing tests for all possible paths

results in 100% statement and branch coverage. This strategy has the biggest potential for the highest number of bugs and errors.

- **Data Flow Testing** explores the flow of data and tries to detect any improper use of data in the program. To achieve that, it is necessary to create the control flow graph first which represents data dependencies and control dependencies between a number of operations.

- **Loop Testing** focuses on the validity of loop constructs. 4 types of loops can be identified: simple loops, nested loops, concatenated loops, and unstructured loops. Each of these categories requires a different testing approach.

2.3.6 White Box Testing Strategies

White Box Testing shares multiple strategies with Black Box testing such as integration and regression testing. The main ideas of these strategies described here 2.3.2.4 and 2.3.2.3 stays the same, only the area of focus is different. The White Box testing is focused on the internal structures of the tested item while Black Box testing is focused on the external behavior. However, there are some strategies that are unique to White Box testing such as Unit Testing and Static Analysis, which are described in the following sections.

2.3.6.1 Unit Testing

Unit testing is the process of testing the smallest testable parts of an application, called units. The main idea is to isolate each part of the application and determine if it behaves as expected. This White Box testing method is the first level of software testing, which takes place before other testing methods such as the integration. Typically, it is performed by the developers as they have the best knowledge about the code and internal structures of the application. However, a common practice is to automate the unit tests and run them after each build. To ensure the effectiveness of these tests, various strategies were developed over the years such as: [14]

- **Logic checks:** They verify the correctness of the logic in the code. They ensure that the code behaves as expected and follows the right path through code given a correct, expected input.

- **Boundary checks:** They are focused on the boundaries of input domains and output results. It explores how the application behaves for valid, invalid, and edge-case inputs.

- **Error handling checks:** These tests are designed to verify the behavior of the application when an error occurs. Typically, the tested unit does not crash the whole application and provides a meaningful error message.

- **Object-oriented checks:** Designed to verify the behavior of the object-oriented code. It ensures that the objects are properly created, updated, and deleted by the running code.

Examples of the UT
Since the main topic of this thesis is strongly related to Unit Testing, it is important to provide at least a few simple examples to illustrate the main principles of this testing method. The following examples are written in Python and use the pytest library, which is a popular testing framework for Python. The idea is to test the basic functionality of the individual functions. Keep in mind that the tested units can be not only functions but also classes, methods, or even a whole module. However, the common practice is to keep the units as small as possible to ensure the effectiveness of the tests.

- **Example 1:**

```
    def add(a, b):
        return a + b
```

The unit test for this function could look like this:

```
    def test_add():
        assert add(5, 5) == 10
        assert add(0, 0) == 0
        assert add(-5, -3) == -8
```

- **Example 2:**

```
def divide(a, b):
    if b == 0:
        raise ValueError(``Cannot divide by zero'')
    return a / b
```

The unit test for this function could look like this:

```
    def test_divide():
        assert divide(10, 5) == 2
        assert divide(0, 23) == 0
        with pytest.raises(ValueError):
            divide(5, 0)
```

Benefits of UT
In software development, Unit testing is considered the first line of defense to catch the problems. Properly written unit tests offer numerous benefits such as [15]:

- **Early bug detection:** UT enables developers to discover the bugs early in the development process before the newly developed code is integrated with other parts of the application. That saves a significant amount of time and resources by preventing the bugs from propagating to the whole tested application.
- **Improved code quality:** Unit testing forces the developers to write clean, reliable, and maintainable code by enabling them to identify and fix bugs and design flaws early in the development process.

- **Facilitates refactoring:** UT enables developers to refactor the code with confidence. Since the unit test implementation is independent of the code, the test case can stay the same and verify the external behavior of the tested unit even if the internal structure of the code changes.
- **Documentation:** Unit tests often serve as a form of documentation. They clearly show the expected behavior of each tested unit and provide a clear example of how to use the unit. That can be particularly useful for the new developers who are not familiar with the codebase.
- **Continuous integration:** UT is a crucial part of the continuous integration process as it allows developers to run the tests automatically after each build. The CI process will be described in more detail later in this thesis.

In summary, Unit tests allow developers to catch defects early in the process, therefore reducing costs, improving code quality and making refactoring of the code easier. Besides that, they play a crucial role in the continuous integration process.

Limitations of UT
Despite the undeniable benefits of this type of testing, it is often time-consuming and requires a significant amount of resources. The usage of UT needs to be carefully considered and applied only when the benefits outweigh the costs. Prior to using UT for your project, it is important to consider the following limitations [14]:

- **Time-consuming:** The implementation of new Unit tests typically takes a significant amount of the developer's time.
- **Not suitable for all types of projects:** Unit testing is problematic when testing a certain type of project. An example of such a project is a UI/UX application, where the behavior of the application is difficult to test in isolation.
- **Unconvenient for legacy codebase:** Writing Unit tests for existing legacy code can prove to be difficult or even close to impossible, depending on the quality of the code.
- **Cannot guarantee the functional correctness:** Even if the unit tests are written correctly, they cannot guarantee the functional correctness of the whole application with its business requirements. They only verify the behavior of the tested unit in isolation, not in the context of the whole application. [16]

In conclusion, unit testing is a great way to discover bugs early in the development process, especially today when the CI/CD process is widely used; the idea of automated tests, including unit tests, has become a popular practice. However, it is important to consider their limitations and use them accordingly. Developers should be aware that the usage of unit testing does not guarantee the functional correctness of the whole application and that it is not suitable for all types of projects. It is necessary to combine this method with other

testing methods such as integration testing, system testing, and acceptance testing.

2.3.6.2 Static Code Analysis

Static code analysis is a debugging method that allows the developers to examine the code without executing the program. This approach aims to comprehend the code structure and verify its adherence to industry standards. This white box method is usually performed by automated tools that scan the entire codebase for potential vulnerabilities and defects. The developers should be aware that static code analysis is rather a debugging method than an actual testing method as the code is not executed. However, it is a commonly used quality assurance technique that can be used to identify potential issues in the codebase. [17]

Types of Static Code Analysis
To ensure the effectiveness of the static code analysis, several methods were developed such as[18]:

- **Failure/Fault Analysis:** Focuses on incorrect component behavior and faults invalid components in the codebase. The transformation description of input-output helps pinpoint errors. The model design specification is verified to identify issues in particular scenarios.
- **Data Analysis:** Focuses on the data flow and data usage in the codebase. It identifies potential issues with data handling and helps the developers to properly implement operations that handle the data.
- **Control Analysis:** Focuses on reviewing the control flows in the calling structure and state transition (e.g., functions, subroutines, methods, or processes). It includes analyzing the sequential order of control transfers and creating a graph of the model with junctions and conditional branches represented by nodes.
- **Interface Analysis:** Focuses on the program's UI and integrated security measures. The goal of this analysis is to ensure a seamless user experience by preventing user errors during software navigation.

Benefits of Static Code Analysis
Static code analysis offers numerous benefits such as early bug detection, improved code quality, and code consistency. All of these benefits are achieved by predefined rules and standards that are used to verify the codebase. Besides that it also plays a crucial role in enhancing the security of the code and prevent potential vulnerabilities such as SQL injection, cross-site scripting, and buffer overflow from being exploited. [17]

Limitations of Static Code Analysis
Despite the undeniable benefits of this debugging method, it also comes with some drawbacks. The most significant limitations are [17]:

- **False Positives:** This method often generates a significant number of false positives errors due to the complexity of the codebase and poorly defined rules.
- **Time Consuming:** The process of preparation of the Static Code Analysis can be often time-consuming due to the need to define the rules and standards along with the configuration of the automated tools.
- **Cannot guarantee the functional correctness:** Even if the static code analysis is performed correctly, it cannot guarantee the functional correctness of the tested functions.

In conclusion, static code analysis is a great way to ensure the good quality of the code and enforce the industry standards along with coding best practices and other rules that are defined by the customer or any other authority. Similarly to the UT, the benefit of the early bug detection is one of the main reasons why this method is widely used as a part of the CI/CD process.

However, it is important to keep in mind that this method not only cannot replace the other testing methods but does not even execute the code. Therefore, it cannot guarantee any functional correctness of the tested functions. Besides that, the configuration of the automated tools and the definition of the rules can be often time-consuming and it is important to consider if the effort is worth the benefits for the specific project.

2.3.7 Advantages and Limitations of Whitebox testing

The advantages of White Box testing can be easily recognized in the description of the individual techniques and strategies. Especially in the section 2.3.6.1. However, this section provides a summary of the most significant advantages and limitations. This summary helps the reader to understand the analysis performed in the following chapters.

Advantages
The most significant advantages of White Box testing are[19]:

- **Early bug detection:** Detecting the bugs early in the development process saves a significant amount of time and resources.
- **Full Code Pathway Coverage:** Full access to the source code allows testing all the possible paths and conditions in the codebase. Including error handling, resource dependencies and additional code logic.
- **Improved code quality:** All the White Box testing techniques are designed to ensure better code quality including removing dead code, improving the code structure, and enforcing standards.
- **Improved security:** The complete control over the source code allows the developers to test all the potential vulnerabilities and security issues, that are often impossible to test with other testing methods. (e.g., buffer overflow)

Drawbacks
The usage of White Box testing has become a popular practice in software development. However, the project managers and developers need to be aware of its limitations and use it accordingly. The most significant drawbacks that should be taken into consideration are [19]:

- **Time-consuming:** This testing technique is often time-consuming for the developers and requires a significant amount of resources. The project manager should carefully consider if it is possible to design the white box tests for the specific project in the given time frame.
- **Difficult to scale:** The test design and implementation require a strong understanding of the source code, its architecture, internal structure, and target systems.
- **Difficult to maintain:** Since specialized tools and frameworks are often used for this testing method, it might be difficult and expensive to maintain the test suite in the long term.
- **Not suitable for all types of projects:** The automated White Box testing is not suitable for projects with a frequently changing codebase as the tests need to be updated along with the reworked code.

Conclusion
In conclusion, White Box testing is a great way to ensure the quality of the code, detect the bugs early in the development process, and enforce coding standards. That allows the developers to create a reliable and reusable code that is easy to maintain. That is crucial for long-term success as it reduces the costs and time needed for future development.

However, some drawbacks need to be considered by the project managers and developers. The most significant is the time-consuming nature of this testing method. The design and implementation of the white box tests require a deep understanding of the source code, its architecture, and target systems along with specialized tools and frameworks. Besides that, it needs to be carefully considered if the type of the project is suitable for this testing method. If the codebase is frequently changing, the maintenance of the test suite can easily become expensive and time-consuming as the tests need to be updated along with the reworked code.

Despite that, for the majority of the projects, the benefits outweigh the costs and the usage of White Box testing has become a popular practice in software development including the CI/CD process.

2.3.8 Gray box testing

As the name indicates, gray box testing combines black box and white box testing techniques. This method increases the testing coverage by allowing the testers to focus on all layers of the tested system. The following section focuses on the details of the two previously mentioned testing methods combined and the advantages and drawbacks of this approach.

In gray box testing, the tester must possess partial knowledge of the internal structure and algorithms of the tested system. This knowledge is essential for

designing test cases similar to white box testing. However, once the test case is designed, it is performed in the exposed interfaces as in black box testing.

This allows the tester to stimulate the system from the outside, while also being able to verify the internal state of the system. This method is particularly useful for complex systems, where the internal state of the system is not easily accessible. [20]

Gray box techniques

In Gray Box testing, a tester must have a solid grasp of internal data structures and algorithms to design effective test cases. Several techniques can be employed to implement thorough test cases, such as [21]:

- **Matrix Testing:** Examines business and technical risks defined by developers in software programs by analyzing variables with inherent technical and business risks.
- **Pattern Testing:** Analyzes previous defects to determine the causes of failure and proactively designs test cases based on the defect analysis template.
- **Orthogonal Array Testing:** A black box testing technique that involves test data with numerous permutations and combinations, preferred for maximum coverage with few test cases and large test data.
- **Regression Testing:** Verifies that software changes or new functionalities do not impact existing system functioning, ensuring that fixing defects does not affect other software functionalities.
- **State Transition Testing:** Applied to systems with various states, test cases are designed to ensure the correct handling of state transitions.
- **Testing Decision Tables:** Uses decision tables to organize and condense complex business rules, generating test cases covering multiple input conditions and expected results.
- **Testing APIs:** Gray box testing, focusing on testing system interfaces (APIs), ensures acceptance of various input formats and proper operation.
- **Data Flow Testing:** Analyzes the flow of data through the system, creating test cases to examine data pathways and identify potential problems with data handling and processing.

Advantages and drawbacks

Gray box testing combines the advantages of both black box and white box testing. It allows the tester to focus on the internal structure of the system, while also being able to verify the system's behavior from the outside. This approach is particularly useful for complex systems, where the internal state of the system is typically not easily accessible to the tester. Besides that, gray box testing brings the following advantages [21][22]:

- **Non-intrusive:** The tested system is not altered or modified during the testing process.

- **Unbiased Testing:** Avoids the bias of the developers, as the testers are not involved in the development process.
- **Testing from the user's perspective:** Gray box testing allows the testers to focus on the user's perspective, while also checking the internal behavior of the tested system.
- **Intelligent test authoring:** Testers can design test cases based on the internal structure of the system.

Despite the undeniable advantages of gray box testing, it also has various drawbacks and limitations that a designer of a testing strategy should be aware of. The most significant drawbacks are [21][22]:

- **Less comprehensive:** Due to limited access to the internal structure and code, testers may not be able to identify critical vulnerabilities.
- **Defect Identification:** It is difficult to associate defects with their root causes in distributed systems.
- **Inconvenient for algorithm testing** Testing algorithms is challenging due to the unavailability of complete access to the underlying logic.
- **Test cases difficult to design:** Designing test cases is a complex and time-consuming task due to the need for a practical understanding of the internal structure of the system.

In conclusion, gray box testing is an effective method for testing complex systems from the user's perspective, while also being able to verify the internal state of the tested system. However, it is crucial to be aware that this method can't replace the other testing methods, and should be used in combination with other testing techniques.

2.4 I/O module development

This section focuses on the I/O module development process in the Company. It introduces the reader to details of how the I/O modules are developed and tested. Besides that, this section serves as a background for an analysis performed and described later in this thesis. Particular focus will be given to the implementation and testing as these are closely connected to the topic of this thesis. All the information in this section was verified by the supervisor of this thesis and the Company's documentation.

2.4.1 Project delivery

The development of I/O modules is divided into several phases and in concept follows the PLM process described in the section 2.2. However, there are some unique aspects as the development of I/O modules is complex and requires a combination of hardware and software development which also brings special requirements for project planning.

Project initiation phase
The project initiation phase in The Company is usually called the "Kick-off" phase. This phase is similar to the standard project initiation phase described earlier in this thesis. Therefore, it will be described only briefly.

The main goal of this phase is to define the project scope, objectives, and deliverables. That includes a process called "requirements engineering" which is a process of defining and documenting the requirements. The main product of this process is a document called "Requirements Specification". Once the requirements are defined, and the target values are set, the requirements specification is followed by a "Feature Specification" documentation which consists of a detailed description of the required features and their behavior. This document serves as a reference for the developers and testers during the execution phase.

In summary, the main goal of this phase in the Company is to define the HW and SW requirements for the product and appoint responsible people for the project who then appoint the project team. Besides that, a series of documents are created that serve as a reference for all parties interested in the project - developers, testers, project managers, and other stakeholders.

Project planning phase
The project planning phase in the Company is no different from the standard described earlier in this thesis. During this phase, the project manager appoints the project team and creates a project plan. Along with that, it is mandatory to adjust the plan to the specific requirements that come with the HW design and development. Along with that, appropriate equipment and tools need to be ordered in advance as it often takes months to get the required HW. The project plan also includes soft and hard deadlines for the testing phases as some of the testing requires to be performed by external entities, which often takes a lot of time and needs to be planned.

In summary, the project manager creates a project plan that in concept follows the standard project planning described in the section 2.2.1.2. The project plan needs to be adjusted to the specific requirements of the I/O module development, such as:

- Long delivery times for required HW
- External testing entities
- Frequent cooperation of HW and SW developers

Execution phase
The execution phase starts once the project plan is approved and the project team is appointed. This phase could be divided into three parts:

- **Preparation phase:** As many projects in the Company consist of improving already existing products, the preparation phase necessary for HW and SW developers to get familiar with the product and its requirements. This phase usually takes a few weeks and takes place while the required HW is being delivered.

- **Implementation phase:** The development is divided into two-week sprints, which are followed by a retrospective meeting and a planning meeting as mentioned above. The details of the implementation process are described later in the section 2.4.2.

- **Testing & Verification phase:** Even though the testing is performed during the whole development process, there is a specific phase that follows the implementation phase where the module is tested on different levels. During this phase, the developers are reassigned to other projects, however, if a bug is found during the testing, they need to be reassigned back to the original project which is typically costly and time-consuming.

The development is closely monitored and controlled by the project manager who is responsible that the developers follow the project plan and meet the deadlines set in the project plan. This corresponds to the standard Project Monitoring and Controlling phase of the PLM process.

Project Closure
This phase is similar to the standard project closure phase described in the section 2.2.1.5. Besides the standard activities, it typically includes steps necessary for obtaining the required certification for the new product to be introduced to the market. That means creating a detailed documentation and user manual for the product, that serves as a reference for the independent certification authorities. Detailed documentation is necessary for future development as it serves as it is a reference for the developers and testers who might work on the same product in the future.

Even though no development is performed during this phase, it is crucial for the long-term success of the product. Any missing documentation or unresolved technical debt can cause problems in future development and lead to costly delays.

In summary, besides the standard activities, the project manager is responsible for creating a comprehensive documentation and user manual that allows the Company's customers to improve their products in the future and obtain the required certifications to introduce the product to the market.

2.4.2 Implementation process

This section describes the development process of the I/O modules in the Company, which takes place during the execution phase of the project. It introduces the reader to the workflow that developers follow during the development as well as the verification processes that are designed to ensure the quality of the code and the comprehensive fulfillment of all specified requirements.

The Company uses an agile feature-driven development methodology with a Scrum framework. In summary, the development is divided into two two-week sprints, which are followed by a retrospective meeting where the team evaluates the sprint and plans the next one. Besides that, the team has a bi-weekly stand-up meeting where the team discusses the progress and the problems they are facing.

The I/O module development is unique as it usually requires specialized hardware that is often developed in parallel with the software. And even if the

hardware is already developed, it is often unavailable. This is quite challenging for the developers, testers and project managers as it requires a lot of planning and coordination. To find an optimal balance between the costly testing on the real hardware and the need for frequent testing to ensure the quality of the code, the Company implements the following measures:

- **Automated testing in CI/CD pipeline:** The code is tested in the CI/CD pipeline after each significant change is made. (White box testing)
- **Integration testing:** Integration testing is performed on the real HW after each internal release (Gray box testing)
- **Regression testing:** Regression testing is performed after each internal release to ensure that the new features do not break the existing ones.
- **Code reviews:** After a significant change is made, the code needs to be reviewed by at least one developer from your team and one developer from another team to ensure that all the teams follow similar code standards and guidelines.

It is a common practice in the Company to create comprehensive White Box tests in the CI/CD pipeline as it allows to catch the majority of bugs before the costly and time-consuming integration testing on the real HW takes place.

2.4.3 Testing of I/O modules

This subsection introduces the reader to the testing of the I/O modules in the Company and serves as a background for the analysis performed later in this thesis. This section does not aim to explain the technical details of the testing process, but rather to provide a high-level overview and introduce the reader to the terminology.

The test workflow in the Company is shown in the figure 2.8. As can be seen, several levels of testing are performed once the release is ready. Each level corresponds to a different level of development as shown in the figure 2.7. The testing is performed in the following order:

- **Automated testing** Typically created by the developers who are responsible for the release. It includes Unit tests, Static code analysis, and other white box testing techniques.
- **Integration test:** Performed by a test specialist/testers on the real HW. The testers have a comprehensive knowledge of the HW design of the module, however, they do not possess the knowledge of the firmware implementation.
- **System test:** Performed by independent test specialists. The goal is to perform a Black box testing that verifies that the module fulfills all the specified requirements. The System Test always ends with a regression test to ensure that the new features do not break the existing ones.
- **Acceptance test:** Typically performed by the customer or the end-user. The aim is to verify that the module fulfills all the functional and non-functional requirements.

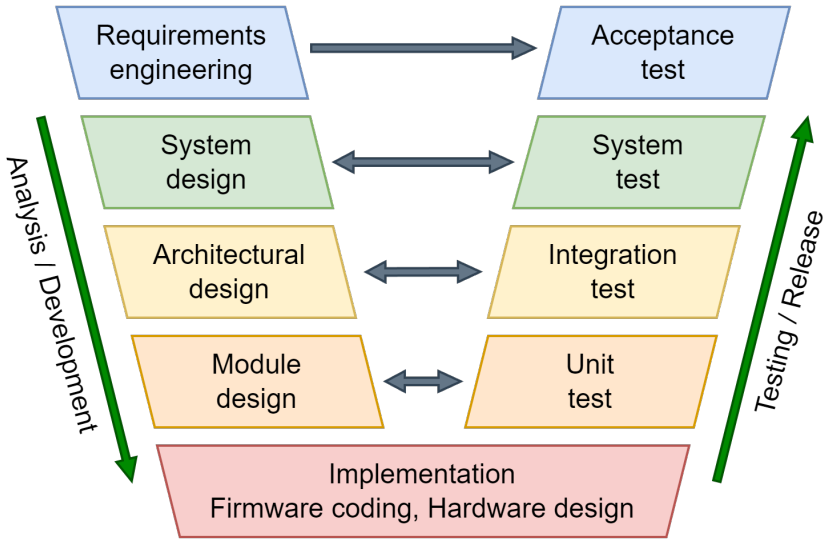

Figure 2.7: Test phases of the I/O module development

To sum it up, the development process includes a series of tests that are performed on different levels. Each level of testing is performed by a different entity and has a different focus. As we go further from the implementation, the test shifts from white box testing to Black box testing as each testing entity knows less and less about the implementation details. It is important to note for future analysis that each testing phase takes a different amount of time and resources. Therefore, it is crucial to make sure the potential bugs are discovered as soon as possible in the testing process.

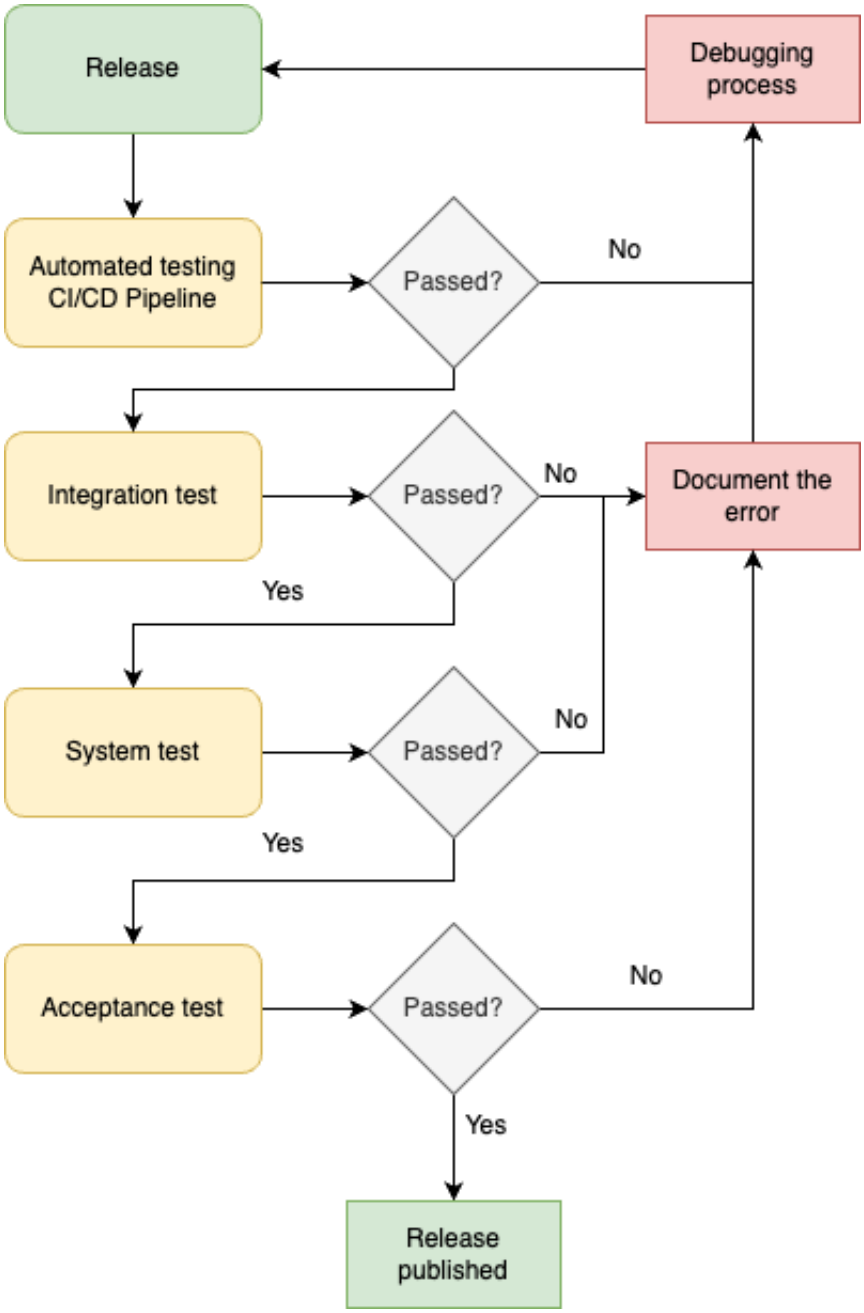

Figure 2.8: Internal release - Test workflow

2.5 Methods of virtualization

This section provides an overview of the most common types of virtualization and introduces the reader to the differences between emulation and virtualization and explore the most convenient virtualization methods for this thesis.

There are many definitions of virtualization in the literature. Therefore this section focuses more on the general principles of virtualization than on the exact definitions. In general, virtualization is a technique that allows more efficient use of hardware resources by creating a virtual version of a device or resources.

Virtualization technology abstracts applications, guest operating systems, or data storage from the underlying hardware or software. Server virtualization, a major application of this technology, utilizes a hypervisor to emulate hardware resources like CPU, memory, and I/O. That offers flexibility, control, and isolation by decoupling software from specific hardware platforms, enabling diverse applications. Despite the fact that the performance may not match that of native hardware, virtualization provides significant benefits for various computing environments. [23]

Several types of virtualization are categorized based on the level of abstraction and the scope of the virtualization. The following list provides the six main areas where virtualization is used[23]:

- **Network Virtualization:** Combines network resources by dividing available bandwidth into independent channels, facilitating real-time assignment to servers or devices, and simplifying network management.
- **Storage Virtualization:** Pools physical storage from multiple devices into a single entity managed from a central console, commonly used in storage area networks (SANs).
- **Server Virtualization:** Masks server resources from users, enabling increased resource sharing and utilization while abstracting server complexities.
- **Data Virtualization:** Abstracts technical data details for broader access and resilience tied to business needs, simplifying data management.
- **Desktop Virtualization:** Virtualizes workstation loads for remote access via thin clients, providing security and portability benefits. Requires accounting for OS licenses and infrastructure.
- **Application Virtualization:** Abstracts application layer from the OS, allowing encapsulated execution independent of the underlying OS, facilitating cross-platform compatibility and isolation.

To sum it up, the term virtualization refers to a process that typically creates an abstraction layer between the physical hardware and the software running on it. This abstraction layer can be represented in different ways, depending on the type of virtualization used. However, the most common is a virtual machine (VM) that partitions existing hardware into multiple isolated environments.

2.5.1 Levels of Virtualization

One of the main reasons for the popularity of virtualization is the flexibility it provides. It can be implemented at different levels, each offering distinct benefits and use cases. As the figure 2.9 shows, there are five levels ordered from the application level to the lowest instruction level. This section provides a brief overview of each level which will later help to choose the most convenient method of virtualization for this thesis.

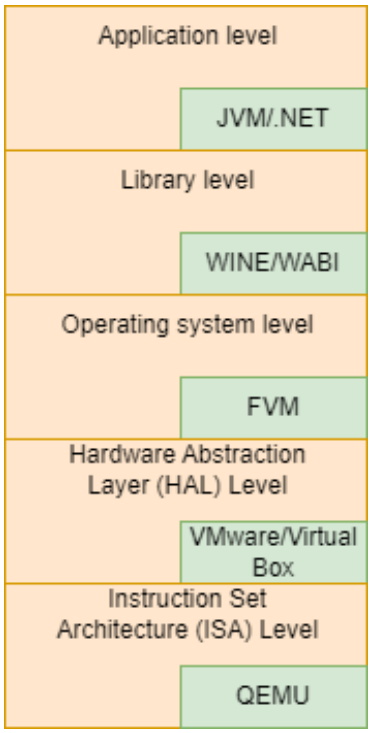

Figure 2.9: Levels of virtualization[24]

Application Level
The highest level of virtualization is at the application level. At this level, the user-level programs and the operating systems are executed in a virtual environment, which behaves like a real machine.

Application virtualization involves creating a layer between an application and the operating system, making the application think it's interacting directly with the OS when it's not. This layer runs subsets of the application virtually without impacting the underlying OS. By consolidating files and registry changes into a single executable file, applications can run smoothly across different devices, including previously incompatible ones. Desktop virtualization, used alongside application virtualization abstracts the physical desktop environment and its associated apps from the end-user device accessing them. [24] [25]

The main benefit of application-level virtualization is that it allows applications to run on different operating systems. This is particularly useful for legacy applications that are no longer supported by modern operating systems or for cross-platform operations such as running Windows applications on a MacOS. Another benefit can be in preventing conflicts with other virtualized applications or the host operating system since each application runs in its own isolated environment. The isolation also allows users to run multiple versions or instances of the same application on the same machine which is useful for many purposes such as testing or development. [25]

Library level
Virtualization at the library level is a technique where software libraries provide an abstraction layer that masks the underlying hardware or software. This abstraction layer is created through library interfaces that allow controlling the communication link between the application and the rest of the system through API hooks. [24]

OS level
OS-level virtualization, often called containerization, creates isolated containers on a single physical server. That is achieved by creating a virtualization layer on top of the operating system which separates the physical machine from its logical structure.

This virtualization layer acts in many ways like a virtual machine manager and allows users to access and run multiple isolated virtual systems on a single physical server. The layer imitates the operating environment, which is recognized on the physical machine, to create a virtual environment for the applications. Applications then run in the containers as if they were running on a dedicated physical machine without recognizing any difference between the container and the physical server. [24]
The main advantages of this level of virtualization are[26]:

- **Resource Efficiency:** OS-level virtualization enhances resource efficiency by eliminating the need to emulate complete hardware environments, thereby reducing resource overhead.
- **High Scalability:** Containers offer high scalability, allowing for quick and easy scaling based on demand, facilitating seamless adaptation to workload changes.
- **Reduced Costs:** OS-level virtualization can significantly lower costs by requiring fewer resources and infrastructure compared to traditional virtual machines.
- **Faster Deployment:** Containers enable rapid deployment, reducing the time needed to launch new applications or update existing ones.
- **Portability:** Containers offer high portability, allowing for effortless migration between environments without necessitating modifications to the underlying application.

On the other hand, there are various disadvantages such as limited isolation between containers, and security risks. Since containers share the same

host operating system, a security breach in one container can potentially affect all others. Moreover, this level of virtualization always requires the installation of virtualization software on the host machine which could be potentially problematic in some cases.

Hardware abstraction layer level

HAL (Hardware Abstraction Layer) virtualization is a technique that uses software component that acts as an interface between the operating system and the hardware. This abstraction layer provides a consistent and a uniform way for software programs to use different physical hardware devices without any knowledge of the specific details of each device.

The main idea behind HAL virtualization is to provide a set of standardized functions and protocols that separate the low-level details of hardware components from the higher-level software programs. Once this set of functions and protocols is defined, the software programs communicate with the hardware devices by using a high-level API, instead of using the device-specific drivers.

The HAL virtualization indeed offers numerous benefits, many of which are particularly advantageous for I/O (Input/Output) module development. The first and foremost advantage is that it significantly simplifies software development by providing a consistent interface for interacting with hardware devices. Therefore, developers can write software programs that are independent of the specific hardware components and do not need to learn about the details of each device they are working with. That is closely related to the second advantage, which is the increased portability of software programs. Since the software programs are not tied to specific hardware components, they can be easily ported to different platforms without any modifications in the code itself.

Besides numerous advantages, the HAL virtualization can be customized for a specific hardware device. This approach is particularly useful for I/O module development since both hardware and firmware are developed in parallel but independently. The hardware developers can focus on designing the hardware while the firmware developers can focus on writing the software since they know the pre-defined interface that the software will use to communicate with the hardware. [24] [27]

ISA level

The lowest level of virtualization is at the instruction set architecture level. This level of virtualization involves creating a virtual environment that emulates the instruction set of a particular hardware architecture. This allows to run software that is designed for a specific hardware architecture on a different hardware with a different instruction set.

To understand the idea behind ISA-level virtualization, it is necessary to understand the term ISA itself. The Instruction Set Architecture defines the interface between the hardware and the software, specifying the functional definition of storage locations such as registers, memory, operations that can be performed on them, and most importantly the definitions of instructions that can be executed. [28]

Virtualization at this level is achieved by creating a virtual component called an emulator that translates the instructions of the guest architecture to

the instructions of the host architecture. The instruction can be of two types: I/O-specific instructions and processor-oriented instructions.

It is a convenient method of virtualization that allows running device-specific software on different hardware. That offers flexibility, control, and isolation by decoupling software from specific hardware platforms. However, the performance is typically extremely slow due to the translation process of the instructions. [24]

2.6 Convenient level of virtualization for this thesis

The main goal of this thesis is to develop a virtual testing framework for industrial I/O modules. To achieve this goal, it is necessary to analyze which level of virtualization is most convenient for this purpose. It is clear that the application level and OS level virtualization are not suitable for this thesis as they are typically not designed for hardware testing and do not provide the necessary level of control over the hardware. Similar arguments can be made for library-level virtualization.

On the other hand, the HAL level and ISA level virtualization are promising candidates for this thesis. They both offer a high level of control over the hardware and allow for the development of custom virtual environments. Besides, they can benefit from an extreme separation of hardware and software, which is crucial for I/O module development since both hardware and firmware are developed in parallel but independently.

The ISA level could be a good choice for this thesis since it allows running device-specific software on different hardware. Using virtualization software such as QEMU, it is possible to emulate the whole instruction set of a particular hardware architecture. However, since this level of virtualization is implemented at the lowest level without any changes in the code of the software, the timing of the software execution is crucial. That would make the emulation process extremely slow and difficult to implement, which is unsuitable for this project. Besides that, the instructions of the guest architecture can differ from project to project, which would make it necessary to implement a new emulator for each project.

However, the HAL-level virtualization would allow running the module firmware on the standard Windows or Linux operating system by implementing the hardware interface that is defined for all I/O modules developed in the company. This approach would allow us to maintain the technology part of the module firmware and replace the hardware-specific parts. This way all the modules from the same family of modules could be tested on the same virtual environment.

Therefore, the HAL level of virtualization is the most convenient method for this thesis. Mainly because it allows to create a customized environment that replaces the hardware-specific parts of the firmware with a customized emulator that allows running the firmware on a standard operating system and gathers the necessary data for testing.

Chapter 3

Analysis and Design

In this chapter, we analyze the currently used testing process in the Company to find room for improvement. Then, we design and describe a virtual testing framework that improves the quality and effectiveness of testing. Special attention is given to the impact of the VT framework on the project delivery process.

3.1 Analysis of the current testing process

This section aims to analyze the current testing process of the I/O modules to identify the rooms for improvement that the VT framework can address. The analysis corresponds to point 3 of the thesis's objectives and is based on the information described in the section 2.4.3, which provides a high level overview of the testing process of the I/O modules in the Company.

3.1.1 Testing time frames and delays

Synchronizing comprehensive testing with module development is a challenging task for a project manager. Each testing phase, described in section 2.4.3, requires a specific amount of time to complete and is dependent on the successful completion of the previous phase. Besides the time required for testing can be prolonged due to the unavailability of the required hardware that is often shared among multiple projects. However, for this initial analysis, let's assume that the hardware is available when needed.

Table 3.1 provides an overview of the time frames for each testing phase along with the estimated time for fixing the bugs discovered during each phase. The time frames are based on the information provided by the Company's testing team and are based on the average duration of the testing phases. Assuming the provided numbers, the total time required for testing can be estimated to be around 4-6 weeks without any bugs discovered during the testing process. However, developing a complex I/O module without any bugs is highly unlikely and unrealistic.

It is crucial to note that discovering bugs means restarting the whole testing process from the beginning. Therefore, the time required for fixing the bugs increases the later the bugs are discovered in the testing process. The following paragraphs describe the process of fixing discovered bugs for each testing

Type of testing	Duration	Delay
Automated tests in CI/CD pipeline	∼1 hour	None
Integration test	1-2 weeks	up to 4 weeks
System test	1-2 weeks	up to 7 weeks
Acceptance test	>2 weeks	>10 weeks

Table 3.1: Time frames for testing phases

phase. Since the acceptance testing does not test the module's algorithms and is focused on other aspects of the module, this phase is excluded from the analysis.

Automated tests in CI/CD pipeline
The automated tests in the CI/CD pipeline are the first line of defense against bugs. The tests are executed automatically after each "git push" to the repository dedicated to the module. This testing phase aims to provide an immediate response to the developer about the newly implemented code. Discovering bugs in this phase usually causes no delays since the developer can fix the bugs immediately. These tests currently include Static code analysis, Unit tests, and newly added virtual tests in the VT framework.

Integration test
The integration tests are performed by specialized testers on the physical hardware. That means there must be a communication between the developers and the testers to schedule the testing and provide feedback.

The first step of this testing phase is to assemble a testing rack with the required hardware. This process can take up to a few days, depending on the availability of the hardware. It is important to realize that assembling the testing rack for one project can block the hardware for other projects.

The second step is to perform the testing according to the test plan. The testers can either perform the tests manually or use automated testing tools. If a bug is discovered, the testers are required to provide a detailed bug report to the developers. Once the bug is reported to the developers and the project managers, a task is created for the following sprint to fix the bug. This means that the bug usually cannot be fixed immediately, as the developers are focused on the tasks planned for the current sprint.

In conclusion, it takes 1-2 weeks to discover the bug in the integration testing phase, then the bug is reported and a task for fixing the bug is planned for the next sprint. Assuming that the bug was discovered in the middle of the current sprint, it will take at least one week before the developers start working on the bug. Even if the bug is fixed in a matter of days, it takes at least 2 weeks to initiate the integration test again. In the worst scenario, it can take up to 4 weeks, which is highly unusual in the Company.

System test
The system test is the final phase of testing that evaluates the module's algorithms. The testing is typically performed by specialized testers in different

locations. This leads to potential delays in communication between the developers, integration testers, and system testers. However, for this analysis, let's assume that the communication is flawless.

A critical aspect of the system test is that it always requires full regression testing of the entire module to pass successfully. This means that even with a tight deadline, there is no way to speed up the testing process.

Let's consider a hypothetical situation where a bug is detected towards the end of the system testing phase. This implies that the module has already passed the integration testing phase, which took approximately two weeks, and the majority of the system testing, which took at least one week. Since the system testing phase is more intricate and the testers are typically less familiar with the internal workings of the module, it generally takes more time to describe the bug and provide a comprehensive bug report to the developers.

Therefore, It takes around 3-4 weeks for developers to discover a bug and plan a task for the next sprint. Assuming the bug can be fixed within one sprint, it takes another 2-3 weeks for the module to be tested. This leads to a total of up to 7 weeks before the module can be sent back to integration testing. Additionally, if we include the required time for integration and system testing, it will take at least 11 weeks to get the module ready for acceptance testing. It's important to note that this timeline assumes no additional bugs are discovered during the second round of testing.

To summarize, identifying bugs during this testing phase can result in significant delays and cause substantial financial losses and other damages to the Company. Hence, the Company has developed a clear strategy to minimize the risk of discovering bugs at such a late stage in the testing process.

Conclusion

The entire process of addressing the bugs that were discovered during the testing stages is explained in Figure 3.1. This figure, along with table 3.1, provides a comprehensive overview of the time needed to test and fix the detected bugs. Based on the information presented, it is important to minimize the risk of finding bugs during the later stages of the testing process. Also, developers require a way to test the module's algorithms before starting another testing round.

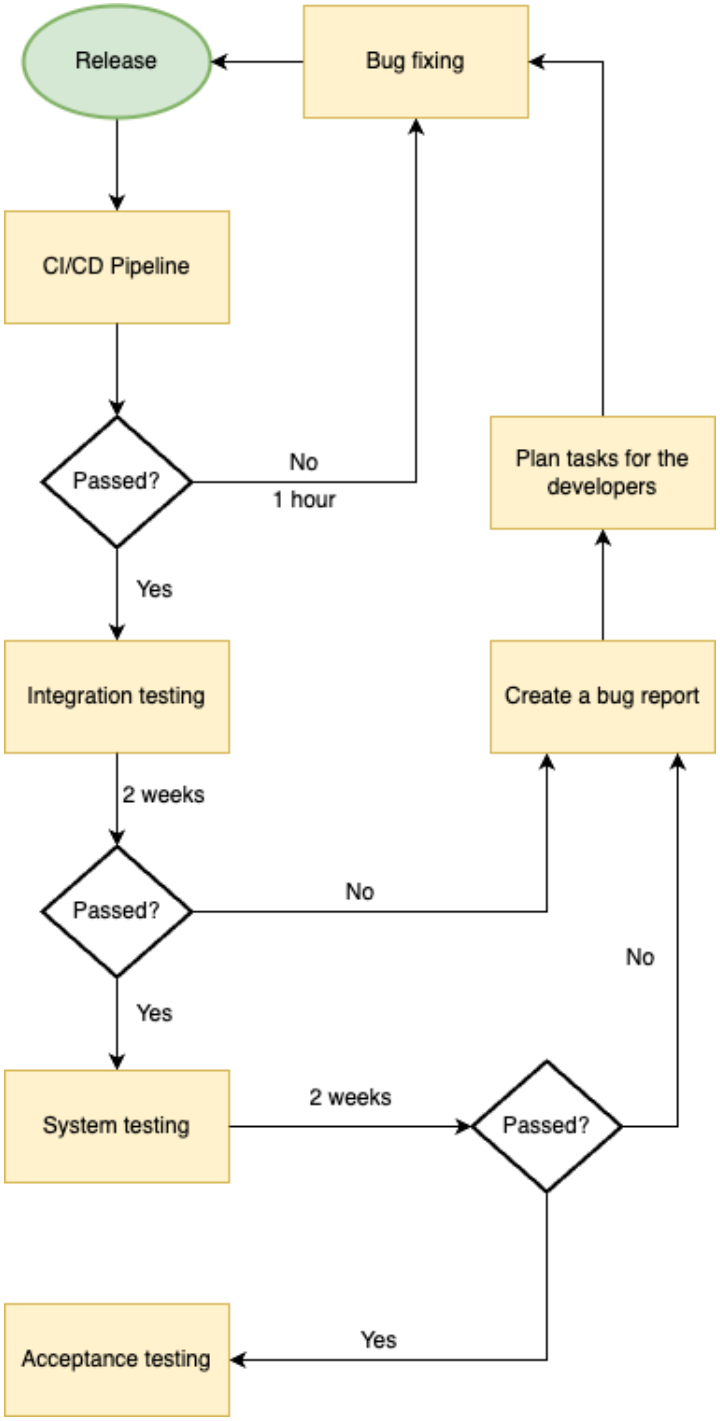

Figure 3.1: Process of fixing bugs discovered during testing phases

3.1.2 Quality of testing

The Company is widely known for its high-quality products that went through a rigorous testing process. It is safe to say that testing quality is crucial for the Company's reputation and its position in the market.

Since the stakeholders are aware of this situation they are willing to invest in improving the testing process. This section aims to analyze the current situation and identify the room for improvement that the VT framework can address.

Ensuring an excellent quality of testing is a challenging task that costs a significant amount of time and financial resources. As we have introduced the testing process and bug-fixing process in the previous sections, we can now go into more technical details.

Development testing
The first level of testing is performed by the developers themselves or by the automated tests in the CI/CD pipeline. These tests are fully designed by the developers and are focused on the module's algorithms.

This testing phase corresponds to typical White box testing and it typically includes Static code analysis and Unit testing. Both of these techniques help developers to identify bugs and ensure the quality of the code. However, it's important to note that these testing techniques can only assess individual parts of the code and not the module as a whole.

That means, the developers have very limited options to verify many types of bugs, such as race conditions, memory leaks, or any other edge cases.

Functional testing
Functional testing is implemented through integration testing of the physical hardware. The integration testers in the Company do not know the internal structure of the module's code and are focused on the module's behavior. That corresponds to typical Black box testing, which means testers cannot verify the module's inner state. That makes it difficult for developers to identify the root cause of the discovered bugs and can lead to significant delays as they cannot replicate the bugs in their development environment.

Having the ability to easily replicate discovered bugs in a development environment, with the option to debug the code and verify the module's internal state, would significantly reduce the time required for bug fixing. However, the goal is to discover the bugs before initiating the integration testing phase.

It is worth noting that the test suites used for integration testing are available to both developers and integration testers. This means that developers are aware of which tests are going to be performed during the integration testing phase. However, during the development phase, they usually face difficulty in debugging their code as they do not have access to the necessary hardware.

System testing
It was stated before, that the goal of the system testing is to verify the behavior of the final product. The testing is performed by a completely independent team that is not involved in the development process. The system testers have a completely independent test approach that is not shared with the developers. The tests are based on the project's requirements and other requirements that

verify the module is compliant with the standards and regulations required for the product to be introduced to the market.

This situation corresponds with the requirement of ensuring unbiased and independent testing. Analysis of the closer details of the system testing is not part of this thesis as it does not aim to improve the system testing.

However, in the previous section 3.1.1, we established the time frames for this testing phase and the time required for fixing the bugs discovered during this phase. The system testing can cause significant delays in the case of newly discovered bugs, the Company strives to improve the quality of Testing in the previous phases and reduce the number of bugs discovered during the system testing.

Conclusion
The Company has a well-established testing process and an excellent reputation for the quality of its products. However, the management is aware that the improvement of the quality of testing on development and integration testing the level can be improved and is willing to invest in such an improvement as it can significantly reduce the number of bugs discovered during the system and acceptance testing.

The following requirements can be derived for the solution to improve the quality of testing:

- **Improve accessibility of testing:** The developers should have the ability to easily replicate discovered bugs or even design their own test cases to verify the module's behavior and internal state.
- **Increase frequency of testing:** Since the required hardware is often shared among multiple projects, it is not always available when needed. Therefore, the development continues without the possibility to verify the functionality of the new code. That significantly increases the time required for fixing identified bugs. Running the tests more frequently would allow developers to always work with verified code and easily recognize new bugs.
- **Extend the test coverage:** It is almost impossible to identify certain types of bugs, such as memory leaks or buffer overflows, with tests run on physical hardware. Those bugs are often identified during late testing phases such as system testing.

These requirements were used as a basis for the design of the VT framework. However, quality of testing is not the only perspective that needs to be considered to create an effective solution that can be adopted by the Company.

3.1.3 Inefficiencies of the testing process

This section aims to summarize the room for improvement in the testing process from the previous sections. Mainly, from the technical and organizational perspective. That corresponds with point 2 of the thesis's assignment and clearly states the inefficiencies that the VT framework can address.

The developers, testers and project managers typically face the problem of discovering the bugs in the later stages of the testing process. That leads to

the necessity of restarting the whole testing process from the beginning and significant delays in the project's timeline as was described in the section 3.1.1. This situation could get even worse by the unavailability of the required hardware that is often shared among multiple projects. In addition to that, there is a significant amount of time at the beginning of the project when the developers and testers are waiting for the hardware to be available and are forced to develop the firmware of the product without the possibility of testing it. That leads to the situation when and complex structure of the firmware is developed with a significant amount of bugs which are time-consuming to discover and fix once the hardware is available.

Therefore, the following problems can be identified:

- **Unavailability of the required hardware:** Hardware is often shared among multiple projects and it takes a significant effort to assemble the testing racks with the required hardware.
- **Lack of feedback for developers:** The firmware developers have very limited options to verify the accessibility of the newly implemented code. That leads to developing complex structures that are time-consuming to debug and fix in case of any problems with their functionality.
- **Discovering bugs in late stages of the testing process:** The early stages of the testing process are not currently designed to identify bugs that occurs less frequently.
- **Lack of Gray box testing:** Discovering a bug is only the first step of the process. The developers need to identify the root cause of the bug by replicating it in their development environment. As you can see in figure 3.2, current testing methods lack Gray box testing, that would allow the developers to easily replicate the discovered bugs while also being able to verify the module's internal state.

Besides the problems mentioned above, there is a significant delay at the beginning of the firmware development. The developers are waiting for the hardware which takes a significant amount of time to be prepared at the beginning of the project. Without the hardware, the testing process can't be The management is aware that the developers often wait weeks before they can start with the firmware development and is willing to invest in the solution that would allow them to start with the development sooner.

The Company's management is aware of these inefficiencies and their significant financial impact and is willing to invest in the solution that would address them.

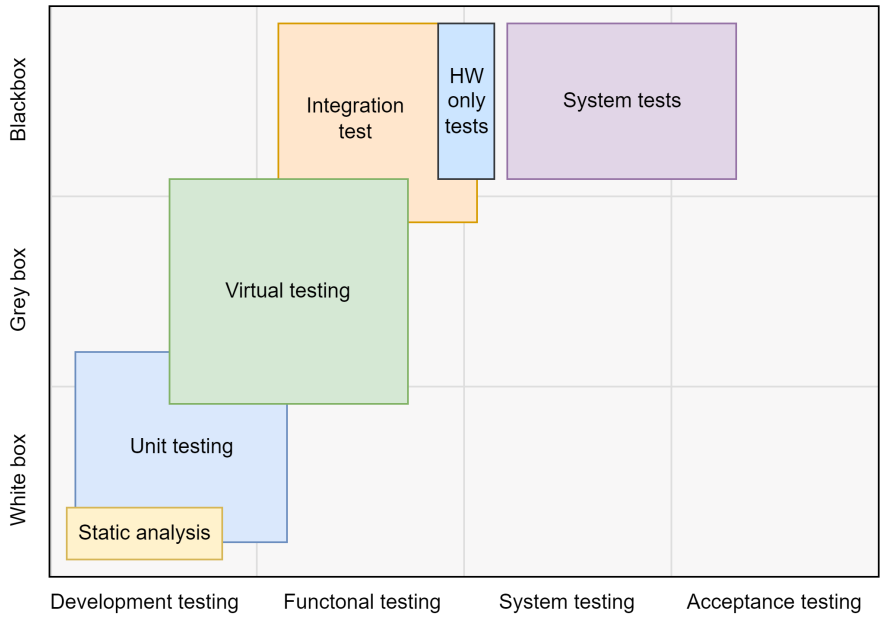

Figure 3.2: Scope of testing

3.2 Requirements engineering for the VT framework

Requirement engineering is a crucial part of the software development process. In this section, the requirements for the VT framework are determined based on the analysis of the current testing process. This section goes into the details of what the Company's management and stakeholders expect from the VT framework and what the framework should provide to the developers and testers.

As each of the involved parties has different expectations from this framework, it took a significant effort to find common ground and define reasonable goals that satisfy all the parties. For a better understanding of the situation in the Company, this section is divided into three subsections, each of which represents the expectations of a different party.

3.2.1 Developer's requirements

The main problem the developers face is the accessibility of testing, especially at the beginning of the project. Their options to verify the functionality of the newly implemented code are limited. The situation is improved by the static code analysis and unit tests running in the CI/CD pipeline, however, any of these methods are not able to verify how the individual components of the tested modules interact with each other.

This situation forces the developers to implement complex structures without verifying their functionality. Once a bug is discovered during the integration testing phase, it takes a significant amount of time to identify where the root cause of the bug is.

Besides that, the only way to replicate the discovered bugs is to assemble their own testing rack. However, running the tests on the physical hardware does not allow the developers to easily look into the module's internal state. Over the years, various methods to log the current state of the modules have been developed, however, they were required to go through the logs manually, which is a time-consuming process that requires an experienced developer.

To sum it up, from the developer's perspective, the VT framework should provide the following features:

- **Easily applicable:** There should be a clear set of instructions on how to set up the virtual tests for new modules. This effort should be minimal and outweighed by the benefits of the virtual tests.
- **Easy bug replication:** The developers should be able to look into the module's internal state while replicating the bugs discovered during the testing process. Ideally, the VT framework should provide a way to run the same tests that were run on the physical hardware.
- **Improved test coverage:** The VT framework should allow developers to verify the behavior of the module in edge cases and under different conditions by providing a way to control the module extremely precisely, which is impossible to achieve with the physical hardware.
- **Integration with the CI/CD pipeline:** Frequent feedback is crucial for the developers to work efficiently. The VT framework should be integrated into the CI/CD pipeline which is automatically executed every time any code is pushed to the repository.

All of these requirements are based on a thorough discussion and analysis with the developers in the Company. Fulfilling them should significantly lower the number of bugs discovered during the later stages of testing and improve the overall quality of the code developed in the Company by providing frequent and precise feedback to the developers.

3.2.2 Tester's requirements

The VT framework is not supposed to replace testing on physical hardware as it is a hard requirement from the customer to test all the products that go to the market on the physical hardware and go through all the testing phases.

Despite this fact, the VT framework could be a valuable tool for the testers, if certain requirements are met. The main requirement from testers is compatibility with the already existing Automation Test Framework. To fully understand this requirement, it is necessary to provide a brief overview of the existing framework and define the word "compatibility" in this context.

The Automation Test Framework was developed in the Company. The framework allows the testers to set up the testing rack and execute a significant part of the test cases automatically or partially automatically. The framework itself is not a part of this thesis, however, the majority of test cases for all modules tested in the Company are written in Python and are using the framework. Therefore, there is an existing requirement from the management to implement the new VT framework in a way that the existing test cases can be reused without any changes.

The compatibility with the existing Automation Test Framework is the crucial requirement for the testers. In case this requirement is not met, the test cases would have to be rewritten, which is a time-consuming process that would prevent the testing team from using the newly designed VT framework.

However, the testers have also expressed a significant interest in the main topic of this thesis and are willing to make adjustments to the automation test framework to make it easier to achieve the desired compatibility.

To sum it up, the main requirement from the testers is compatibility with their existing framework. As the VT framework is supposed to be used primarily by the developers, the testers are willing to adjust their existing framework to achieve better compatibility and reusability of the existing test cases.

3.2.3 Management's requirements

The Company's management needs to ensure that investing in the VT framework will provide a significant return. In addition to the technical requirements, they must consider the financial aspect of the investment and the Company's overall testing strategy.

The testing strategy aims to synchronize the testing process across all Company locations and ensure that test suites implemented at one location are easily reusable at other locations. This minimizes the effort required when a module development is moved from one team to another.

Furthermore, the management aims to create a solution that allows developers and testers to create test cases in a unified way. This prevents each entity from creating its own tests during development, which duplicates the effort required for testing. Such duplication is a significant waste of resources and is not in line with the Company's testing strategy.

However, the most crucial requirement from the management is the quick implementation of the VT framework for new modules. The VT framework should aim to be applicable in the early stages in 2-3 weeks. This requirement would allow the developers to use the time at the beginning of the project more efficiently and start with the firmware development sooner. Even before, the hardware is available.

To summarize the management's requirements, the VT framework should provide the following features:

- **Unified way of creating test cases:** The VT framework should provide a way to create test cases in a unified way to prevent the duplication of effort.
- **Compatibility - locations:** The VT framework should be compatible with the existing solutions to ensure it can be used across all Company locations.
- **Quick implementation:** The VT framework should be applicable in the early stages of the project in 2-3 weeks.
- **Improve quality of testing:** The VT framework should aim to improve the quality of testing on the development and integration testing level by allowing to create tests for various edge cases and conditions that are impossible to achieve with the physical hardware.

- **Test cases reusability:** All the test cases implemented for the physical hardware that does not depend on the hardware behavior itself should be reusable for the virtual testing without any changes in the test cases themselves.

Even though there are multiple requirements from the management, they all have the same goal. Lower the necessary resources for testing while improving its quality. From the management's perspective, the success of the VT framework is determined by the ability to fulfill this goal.

3.2.4 Outcome of the requirements engineering process

The final set of requirements for the VT framework was derived from the requirements of the developers, testers, and the Company's management. Besides that, the requirements were also discussed and approved by the Company's stakeholders who are responsible for financing this project.

After a thorough discussion and analysis, the following requirements were created for the VT framework project:

- **Easily applicable:** The VT framework should be applicable in the early stages of the project in 2-3 weeks and allow the developers to start with the firmware development even before the hardware is available.
- **Compatibility with solutions across all Company locations:** The VT framework should be easily usable with the existing solutions across all Company locations, including the Automation Test Framework in Prague.
- **Integration with the CI/CD pipeline:** Virtual testing should become part of the automated testing process in the CI/CD pipeline, to provide frequent feedback to the developers.
- **Improved quality of testing:** The VT framework should allow developers and testers to control the module extremely precisely and verify its behavior in edge cases and under different conditions that are impossible to achieve with the physical hardware.
- **Provide Gray box testing:** The VT framework should provide a way to verify the module's internal state and replicate the firmware bugs easily.
- **Lower the necessary resources for testing:** The goal of the VT framework is to lower the necessary resources for testing by improving its quality and efficiency.

In addition to the technical requirements, the Company has various policies that apply to all the projects. These policies are not directly related to the VT framework, however, they have to be considered during the implementation of the VT framework. Examples of such policies are the usage of C++ programming language, following coding guidelines, and standard git workflow.

The formal and informal requirements above were discussed with the Company's management and stakeholders and were officially approved. To correspond with the PML methodology used in the Company, the official project documentation was created and the project was officially kicked off.

3.3 Virtual testing framework

The goal of this section is to design a virtual testing framework based on the requirements defined in section 3.2. The section introduces the reader to the architecture of the virtual testing framework and describes how it fulfills the requirements that the stakeholders have defined. Then the focus shifts to the individual components of the solution and how they interact with each other.

3.3.1 Architecture

The main requirement that the architecture of the VT framework must fulfill is compatibility with the existing testing infrastructure. The VT framework must be able to interact with the automated testing frameworks that testers use to run tests on real hardware. The problem is that each location of the Company typically uses a different testing framework.

Therefore, the architecture needs to be modular and flexible. There must be a clear separation between the components that interact with the technology part of the module and the components that handle the testing infrastructure. In addition to that, the testing infrastructure is constantly evolving at each location of the Company.

For these reasons, the architecture of the VT framework is based on a client-server model. The server is responsible for managing the virtual testing environment and the client represented by any testing framework that the Company uses connects to the server to run test cases.

The architecture of the VT framework is composed of three main components:

- **Virtual testing library(DLL):** A layer that is responsible for wrapping the technology part of the module and allowing the code to be compilable and executable on the Windows platform. This component is often referred to as the emulator.
- **Virtual testing app (Server):** An exe application that is able to load the DLL and manage the communication between the emulator and the testing client.
- **Testing client:** Any testing client that is used within the Company and implements the pre-defined interface for the VT framework.

The general overview of the architecture is shown in Figure 3.3, where we can all the components and their interactions. Each of these components is described in more detail in the following sections including the communication between them.

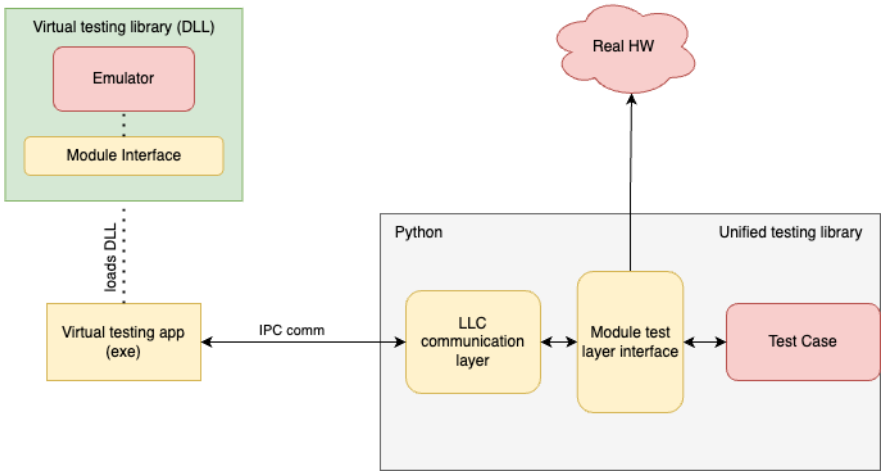

Figure 3.3: Architecture of the VT framework

Module components

Prior to the description of the individual components of the VT framework, it is important to understand the common structure of the modules developed in the Company. The modules are typically divided into two main parts: a technology part and a shared part called BaSy.

What is important to understand is that the BaSy component only provides an interface for hardware control. The technology uses the interface functions to control registers, LEDs, and other hardware components. The BaSy stays the same for the whole family of modules and any change is extremely rare due to compatibility reasons.

The programmable logic is implemented in the technology part of the module. Everything that determines the external and internal behavior of the module is included in this particular component. Since the developers work with the technology component, the VT framework aims to verify its behavior.

Understanding this structure is crucial for future understanding of the VT framework architecture. For the reasons mentioned above, the BaSy component can be modified or replaced by a mock component as it does not affect the behavior of the technology part of the module and allows the emulator to access the internal structures of the module.

On the other hand, there must be no changes required in the technology for the virtual testing. Even a small change would invalidate the results of the tests as it would not be clear whether the module behaves the same way as it would on the real hardware. That is why the technology is encapsulated in the DLL and the emulator accesses its functions and data through pre-defined functions and by implementing the BaSy interface.

To sum it up, the main access point to the technology part of the module is the BaSy interface which originally serves as a communication layer between the technology and the hardware. That ensures that from the perspective of the technology, there is no difference between running on the real hardware and running on the emulator, which is crucial for the validity of the test results.

3.3.2 Virtual testing library

The virtual testing library otherwise called the Emulator is the core component of the VT framework. This part of the framework is responsible for wrapping the technology part of the module and allowing the code to be compilable and executable on the Windows platform. Besides that, the emulator allows the testing client to control the technology part of the module and read data from its inner data structures.

The external behavior of the emulator is clear. It must be able to control the technology part of the module, allow the testers to modify the state of the internal structures and read the data from them. At the same time, the emulator cannot modify the behavior of the code in any way. From the perspective of the module code, there must be no difference between running on the real hardware and running on the emulator, otherwise, the results of the tests would be invalid.

Due to the complexity of this component, it requires its own separate architecture and design. As shown in figure 3.4, the DLL is composed of three main parts:

- **DLL interface:** This part of the DLL is responsible for exposing the functionality of the emulator to the testing client. It is a set of functions that the testing client can call to control the technology part of the module through LLC(Low-Level Commands).
- **Emulator:** This part of the DLL is responsible for wrapping the technology part of the module. The data objects and functions allow control of the inner state of the module and read the data that would normally be stored in the registers and memory of the real hardware.
- **Technology part of the module:** The part of the module that is being tested. This code cannot be modified in any way and must behave the same way as it would on the real hardware. All the changes that need to be done to make the code compilable and executable on the Windows platform need to be done in other parts of the DLL.

DLL interface

The DLL interface is a set of functions that the testing client can call to control the module. Controlling the module means setting the internal state of the module, reading the data from the module and executing the module for a given amount of emulated time.

To fully understand what the DLL interface needs to provide, we need to understand how the testing of any I/O module is typically done. To successfully run a test case, the testing clients require the following functionalities:

- **Start and Initialize:** This function is used to start the module and initialize its internal state to default values.
- **Insert submodule:** The module is often composed of multiple logic submodules. Each submodule is identified by a unique ID. Since each submodule can differ in the size of its data objects, the specific submodule needs to be inserted into an imaginary slot to allow the code to control the validity of the data that is being written or read.

- **Read/Write data record:** All I/O modules are being parametrized through special data objects called data records. The operations of properly writing and reading the data records are specific from other operations and require special functions.

- **Read/Write data objects:** Each module in the family needs its own set of data objects. This interface function allows us to modify them and read their values based on the data object handle defined at the beginning of the communication.

- **Execute Emulation:** The emulation of the module is executed for a given amount of emulated time. This concept is necessary because the processes in the modules are time-dependent and the testers need to access the data at specific points in time.

- **Query for handle:** The handle is a unique identifier for each data object. This function is used by the testing client to get a handle for specific data. This handle is then used to read and write the data into the data object through the "Read/Write data objects" functions.

Through these functions, the testing client can fully control the technology part of the module. This layer is necessary to isolate the testing clients from the logic hidden inside the Virtual testing DLL. This way, any changes to the internal logic of the DLL will not affect the testing clients, which is crucial for the maintainability of the framework.

Emulator
The emulator is the core of the whole VT framework that allows the execution of the technology part of the module without access to the physical hardware. As the figure 3.4 shows, the emulator is composed of three parts:

- **Data objects:** These data objects are used to handle data that are passed to the technology part of the module. There are data objects that are common for all modules in the family such as data records, and data objects that are specific for each module. It is important to note that the data stored in these objects can be only modified by the technology part of the module, not the emulator itself as it would lead to invalid test results.

- **Emulation functions:** This part of the emulator handles the data objects and the execution of the technology part. It is responsible for calling the right technology functions at the right time. This logic needs to be adapted for each module along with the definition of the specific data objects. However, the general rules are shared among all modules in the family.

- **HW emulator:** This part of the emulator is responsible for emulating specific hardware parts of the module such as registers. This part typically implements the BaSy interface and gathers the data that would be handled by the hardware.

Each of these parts has its own specific purpose. The separation of concerns here is crucial for the maintainability of the framework and the ability to adapt the framework to new modules.

To better understand the emulator component, we need to identify what responsibilities it has. The emulator is responsible for the following tasks:

- **Process commands from the DLL interface and execute them accordingly:** The DLL interface only calls the functions that are defined in the emulator. The emulator is responsible for controlling the module by calling the right technology functions or reading the data from the data objects.
- **Implement HW interface for the technology part of the module:** The HW emulator ensures that there are no changes required from the technology code perspective. It is responsible for reading and writing the data that would be stored in the registers or memory of the physical hardware.
- **Data handling:** The Emulator is responsible for handling the data and transferring it between the technology code and DLL interface in the correct format. That is achieved through the pre-defined data objects or other convenient data structures.
- **Call the right technology functions at the right time:** The emulation is always executed for a given amount of emulated time. Since the execution is not real-time, there must be other mechanisms that ensure correct behavior. That is achieved through the complex mechanism called *Event calendar*, which schedules the events and executes them when the time comes.

To sum it up, the emulator is responsible for all the logic that is required to control the technology part of the module. module and handle the data. It consists of three main parts: data objects, emulation functions and HW emulator, where each part has its own specific purpose. That is crucial for the maintainability of the framework and the ability to adapt the framework to new modules. This section only provides a high-level overview of this component as it is an important part of the software design. The actual details of the implementation will be described in the chapter 4.

Technology code

The technology code is a part of the DLL, however it cannot be modified in any way. This is the code that is being tested and it is the same code that would be executed on the physical hardware. All the other parts of the DLL are responsible for wrapping this code to make it compilable, executable and controllable without the need for the physical hardware.

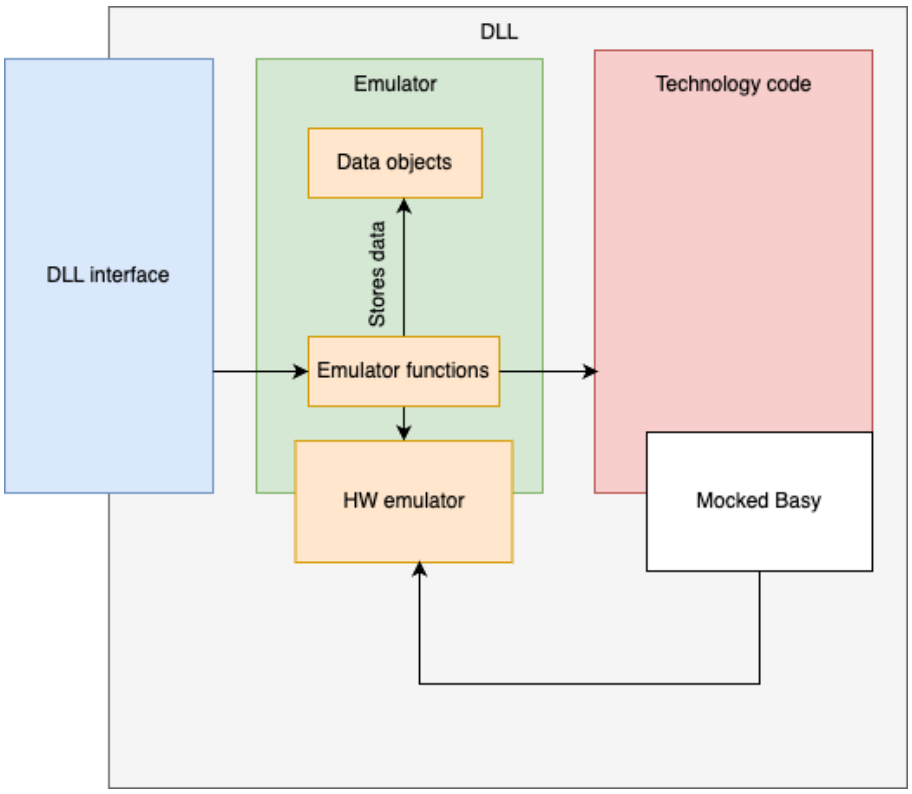

Figure 3.4: Architecture of the Virtual testing library(DLL)

3.3.3 Server application

As the name of this component suggests, the server application is part of the VT framework that is responsible for managing the communication between the testing client and the virtual testing library. After further analysis, the following requirements for this component have been identified:

- **Communication with the testing client:** The server application needs to handle the interprocess communication with the client. This means that the server application must be able to receive the commands from the testing client and send the responses back to it.
- **DLL handling:** The application must be able to load/unload the DLL and call its interface functions. Besides that, it needs to hold the current context of the DLL and be able to reset it to the initial state if required.
- **Logging:** The server application must be able to log the communication between the testing client and the virtual testing library.
- **Error handling:** The server application must be able to handle the errors that occur during the communication. This means that the server application must be able to detect the errors and throw appropriate exceptions.

The need for a standalone server application comes from the fact that there is a possibility of creating other testing libraries for different types of modules. There is no reason to create a different server application for each library as the communication protocol is supposed to stay the same for all of them.

3.3.4 Testing client

The testing client is part of the VT framework, however, the development of the testing clients is not in the scope of this thesis. As we know from the requirements for this project, the goal is to create a framework that is compatible with the existing testing clients in the Company.

However, there are following requirements that the testing client must fulfill in order to be compatible with the VT framework:

- **Implements the LLC communication protocol:** The client needs to implement the LLC communication protocol described in section 3.3.5.
- **Test cases need to stay in the same format:** There must be no changes in the format of the test cases. The same scripts that are used to run the tests on the real hardware must be used to run the tests in the virtual environment. All the changes that need to be done must be done in the backend library of the client.
- **Error handling:** There must be a special layer in the client that interprets the errors that are reported by the server application.
- **Data object preparation:** The client must support the pre-defined data objects to properly interpret the data from the virtual testing library.

Given these requirements, there is a need for a clear separation of concerns to multiple layers in the testing client. The proposed architecture of the testing client is shown in Figure 3.5. The goal of this design is to create a client where the test cases stay the same for both the real hardware and the virtual environment. That is achieved through the proper separation of the test cases, the backend library and the communication layer. As you can see, the test cases are injected into the backend library which then chooses whether to use the implementation for the real hardware or the virtual testing library.

It is important to note, that at all levels the interfaces stay the same. The only thing that changes is the implementation is hidden behind those interfaces, which is crucial for the maintainability and reusability of the solution.

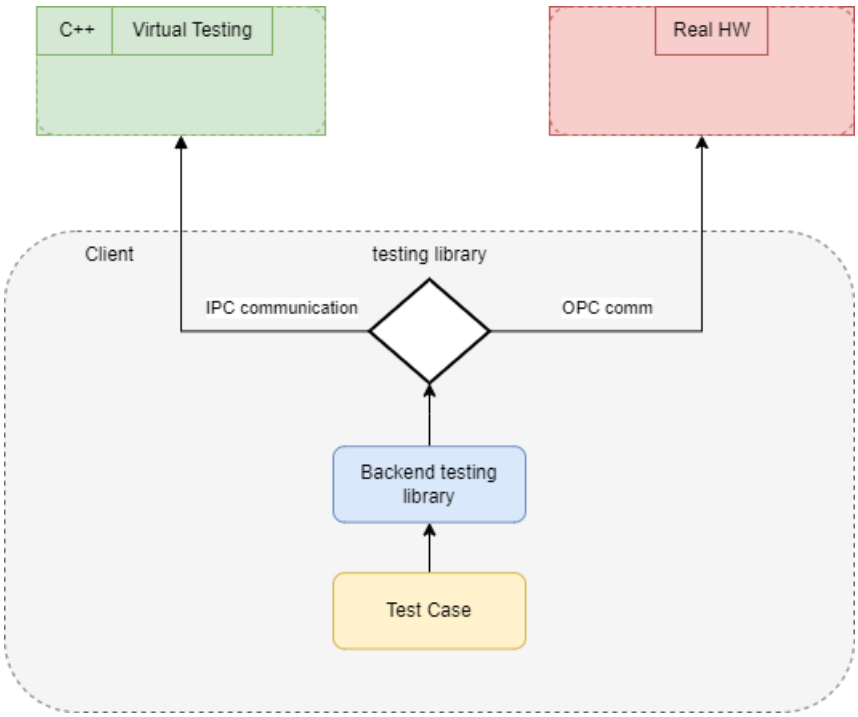

Figure 3.5: Architecture of the testing client

3.3.5 Communication & Interfaces

The communication between the testing clients, server application and the virtual testing library was mentioned in the previous sections. This section focuses on the communication protocol and the interfaces that are used to separate the individual components as much as possible.

The VT framework proposes a communication protocol based on LLC (Low-Level Commands). The protocol is designed to be simple, easy to implement and platform-independent. That means that the way how the commands are transmitted does not influence the format of the data.

As mentioned in the previous sections, the architecture of the VT framework is based on the client-server model. That means that the clients send commands to the server, which then processes the commands and responds to the client. Both commands and responses are transferred in the form of frames, which are categorized into two types: rx_frame and tx_frame.

As it is shown in the figure 3.6, the frames are divided into three separate parts:

- **Header:** The header contains information necessary for processing the data in the payload such as its length.
- **Function code:** The function code is used to determine which VT library function should be called to process the command.

- **Data:** The data should be processed by the technology code. Typically in the form of a byte array.

Each part of the frame is mandatory for any valid frame. If there is any inconsistency in the frame, the server must respond with an appropriate error code. Once the command frame is received and processed by the server and the VT library, the server responds with a response frame.

From the high-level perspective, the format of the response frame is very similar, except for the function code, which would be redundant. If we go into more detail, the data included in the header of the response frame are different. The most important part of the header is the status code, which is used to determine whether the command was processed successfully or not. The client should always check the status code before going through the data in the payload, as the data in the payload might not be valid. In addition to the status code, the header contains the length of the payload, which allows the client to validate the data prior to processing it.

To sum it up, the client and server communication is based on the LLC protocol. Both commands and responses are transferred in the form of frames, which slightly differ for each direction of the communication. The header of the frame contains the information necessary for processing the data in the payload such as - message length, status code, and flags that can be used for additional functionalities. The function code is used to determine which VT library function should be called to process the command as the server app should not add any additional logic to the command processing. The protocol is designed to be simple, easy to implement and platform-independent, which is necessary for the maintainability and reusability of the solution for all locations of the Company.

Figure 3.6: Low-level command

3.4 VT Framework's Impact from a Project Management Perspective

This section explores the impact of the VT framework on the project delivery process and provides valuable insights from a project management perspective. It presents a comprehensive analysis that project managers within the Company should carefully consider when deciding about the adoption of the VT framework for their projects. The analysis encompasses crucial factors such as the enhanced quality of testing, reducing the risk of project delays, and the financial impact.

3.4.1 Project planning impact and risk reduction

One of the biggest responsibilities of a project manager is to ensure that the project is delivered on time and within the budget. That requires careful plan-

ning, which is often a complex and challenging task, especially if there are too many unknowns. In addition to that, the project manager needs to properly manage their resources and appropriately distribute the workload among the team members. This subsection aims to analyze the impact of the VT framework on the project planning process, resources management, and risk reduction. The analysis is based on the information described in the previous sections.

Properly managing the resources assigned to the project is the key responsibility of the project manager. The improvement in this area can lead to a significant reduction in the costs and time required to deliver the project. The Company's management is quite aware of this fact and has been actively looking for ways to support their project managers in this area.

The most significant constraint that the project manager faces is the limited availability of the testing resources that can be allocated to the project at any given time. That often leads to situations where the developers are not able to get feedback on their work in time and are forced to wait for the testing resources to become available. In addition to that, the project managers often cannot predict when the module will come back from the system testing phase, which makes it difficult to plan the next steps in the project.

To sum it up, project managers typically face the following issues:

- **Limited availability of testing resources:** The testing rack needs to be assembled for the module to be tested, which takes time and is often not available when needed.
- **Unpredictability of the testing process:** The project managers often cannot predict if the module will contain more bugs that need to be fixed. That makes it often difficult to plan the next steps.
- **Accesibility of testing:** The developers often cannot run the tests on their own, or replicate the bugs that were reported by the testers.

All of these issues make it difficult for the project manager to properly allocate their most valuable resources - the employee's time. Currently, the common practice in the Company is to allocate from 10% to 20% of the project time as a buffer for the additional testing that might be required. However, this approach is not always effective and forces the project manager to allocate more time than necessary.

The VT framework aims to address these problems by providing the developers with a way to create and run their own test cases, easily and quickly replicate a majority of firmware bugs discovered during the integration and system testing, and get almost immediate feedback on their work. By lowering the number of bugs that are discovered during the late stages of the testing process, the unpredictability of the project is reduced, which allows the project manager to allocate less resources.

In conclusion, by adaptation of the VT framework, the project managers can expect:

- **Reduced unpredictability of the testing process:** Improved quality of testing at the developer level will lead to a lower number of bugs discovered during the later testing stages, which will lower the risk of additional testing time required.

- **Easier access to testing for the developers:** The requirements for the VT framework clearly state, that the developers should be able to run all of the tests from the official test suite if there are no hardware dependencies. That helps them continuously test their work and get immediate feedback, which also helps them to find a potential bug faster.
- **Frequent feedback by automated CI/CD pipeline:** The developers typically code in small increments and push their changes to the repository multiple times a day. VT framework allows the developers to get feedback on their work almost immediately, which helps them to find and fix the bugs faster. Besides that, the developers are always able to test their code to some extent, even if the testing rack is not available.
- **Lower probability of project delays:** Improved accessibility of testing for the developers and allowing them to create and run their own test cases for various edge cases, leads to a lower probability of project delays caused by the testing process.
- **Lower probability of project budget overrun:** By reducing the unpredictability of the testing process, and lowering the number of bugs discovered during the later stages of the project, the project manager can more accurately allocate the resources for the project with a lower risk of budget overrun.

The reasons mentioned above make the VT framework a valuable tool for project managers. Even if we do not consider the money saved by reducing the number of testing rounds required, the improved predictability of the project and lowering the risk of project delays is a significant benefit for the Company's management a customers.

3.4.2 Unified testing strategy

The Company is a global organization with an extensive network of development and testing teams located in various locations around the world. To avoid the duplication of work at different locations and teams, the Company defined a unified testing strategy that all of the teams should follow. The strategy does not enforce the use of specific tools, frameworks and programming languages, but rather defines the requirements that new solutions should fulfill to be considered as a part of the unified testing strategy. That ensures that the new solutions are compatible with the existing testing frameworks and can be reused by the teams at different locations.

The management is actively looking for solutions that are not useable at a single location, as it does not bring benefits in the long run. For this reason, the Company's management and stakeholders need to carefully consider whether the VT framework aligns with these rules and be reused by multiple teams.

After analyzing the requirements for the VT framework, it is clear that the Emulator DLL can be used by any testing client that implements the defined communication protocol. The benefit of this approach is that it does not matter how the testing client is implemented because the low-level commands need to stay the same by definition. In other words, the testing client can be

implemented in any programming language such as Python or C#, but that does not affect the compatibility with the Emulator DLL itself.

Despite the fact there is required an additional effort to implement the pre-defined interface to use the VT framework, the management decided that it was a reasonable trade-off as the implementation should be straightforward and only a one-time effort for each location.

In conclusion, the management of the Company decided that the VT framework aligns with the unified testing strategy as it can be relatively easily reused by the teams at different locations. The additional effort required to implement the pre-defined interface is considered a relatively small and most importantly a one-time effort.

3.4.3 Financial impact

The analysis of the financial impact of the VT framework was crucial for the Company's management, stakeholders and project managers. The decision to invest in the framework was based on the potential financial benefits that could be achieved by adopting this solution. In addition to that, this subsection explores the decision-making process that the project managers within the Company should consider when deciding about the adoption of the VT framework for their projects.

In order to provide a comprehensive analysis, the project's expenses need to be explored to identify where the biggest potential room for improvement lies. The analysis is based on the information provided by the Company's management and project managers. Once the expenses are identified and analyzed, the focus shifts to the potential financial benefits and the return on investment that the VT framework can bring to the majority of the projects within the Company.

Budget distribution

The Company has a significant amount of projects that are developed in parallel. The size of the projects varies from small projects that are developed by a small team of developers to large and complex projects that require a significant amount of resources and time to be developed.

The company keeps a close eye on the expenses of all its projects. After analyzing the data, it was found that three main expenses are common to all projects, regardless of their size. Therefore, this analysis focuses on the following expense categories:

- **Employees:** The salaries of the developers, testers, and project managers.
- **Hardware:** The cost of the hardware that is required for the module development.
- **Licenses:** The cost of the software licenses that are required for the module development.

The budget distribution for the projects is shown in the figure 3.7 based on the data provided by the Company's management. The numbers are based on the average budget distribution as the numbers slightly vary from project

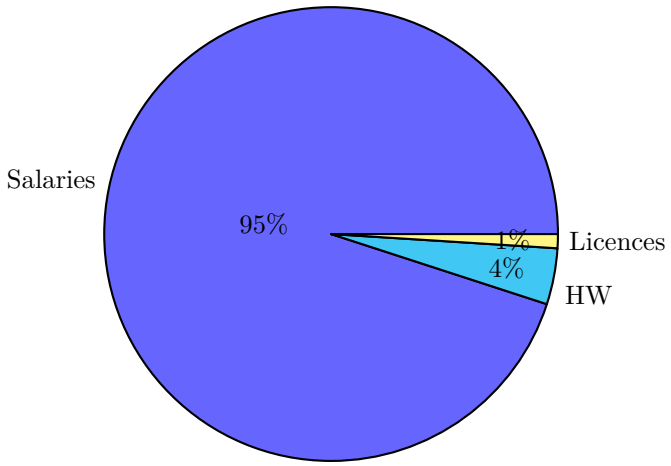

Figure 3.7: Budget distribution

to project. The biggest part of the budget is allocated to the salaries of the employees which is typically from 90% to 98% of the total budget. That means the most expensive part of the project is the time of the employees who are working on the project. Any improvement that can reduce the time required for the project's development or testing significantly reduces the project's expenses.

Even if the VT framework could allow lower expenses on the hardware or licenses, the impact on the total budget would be negligible and would not justify the investment in the framework. Therefore, they can be excluded from the analysis and the focus should be on the potential reduction of employees time required for finishing the project.

Sample Project definition

To explore the financial potential of the VT framework for a project, a hypothetical project will be created. Using the project, we will go through the process of identifying the potential financial benefits and the return on investment that should be considered by the project managers when deciding about the adoption of the VT framework for their projects.

The project is based on the average and most common type of project in the Company. Since we already established that the expenses for the hardware and licenses are negligible for the VT framework, the focus will be only on the time of the employees and therefore the efficiency of the project's development and testing process.

The project consists of 3 developers, two integration testers, and one project manager. Even though the salaries of each employee vary, we can estimate the average expenses per employee to be 50 €. Therefore, from the stakeholder's perspective, efficiency is based purely on the time of the employees, not on the cost of their actual salaries. It is important to note that the management and stakeholders are interested in reducing the overall time and the stability and predictability of the project's timeline.

Initial cost
The adoption of the VT framework requires an initial investment at the beginning of the project. The preparation of the virtual testing takes 2-3 weeks of the developer's time regardless of the project's size. Knowing the hourly rate of the developers, and the 40-hour work week we can estimate the initial cost of the VT framework adoption to be from **4,000 € to 6,000 €** per project.

This estimation gives the project managers a clear understanding of how much time and money the VT framework requires to save during the project's development and testing process. On the other hand, it is important to realize that the adoption of the VT framework can be started before the hardware is available and the development of the firmware can start. That means for certain projects the adoption of the VT framework is implemented while the developers are already assigned to the project, however, they cannot fully start with the firmware development and the Company is losing money due to bad planning or unforeseen delays in the hardware preparation.

For the purpose of this analysis, we will assume the hardware is available when needed and the initial cost of the VT framework adoption is in the range stated above. That being said, the usage of the VT framework needs to save at least 80-120 hours of the employee's time to be considered a good investment for the project.

Bug related expenses
The goal of the VT framework is to reduce the number of bugs discovered during the later stages of the testing process and therefore reduce the project delays and the time required to fix them. Knowing the hourly rate of the employees, we can estimate how much a discovered bug costs in each phase. The estimation is based on the information provided in the section 3.1.1, which describes the process of fixing the bugs discovered during each testing phase.

A bug discovered during the Automated tests in the CI/CD pipeline during the feature development typically has no additional cost. Since the developers receive immediate feedback, they can fix it inside the scope of the current sprint and the time assigned to the implementation of this feature.

Bugs - Integration testing
The situation is different for the bugs discovered during the integration testing. When bugs are found during integration testing on physical hardware, the developer responsible for fixing them needs to release the feature and move on to other tasks planned for the current sprint. This means that fixing the bugs is an additional cost in terms of time and may delay other planned tasks. Besides that, any time there is a bug discovered during the integration testing, at least all test suits connected to the bug need to be tested again. The table 3.1 clearly states that the time required for integration testing is from 1 to 2 weeks. However, the actual time that the testers spend on the testing is only a part of the waiting period. According to the project manager and the team leader of the integration testers in the company, the actual time required for the testing is 3-5 working days, which is around 24-40 hours. The rest of the time is just waiting for the hardware or testers who are busy performing the tests for other projects. Therefore, one round of integration testing performed by 2 testers costs from 2400 € to 4000 € worth of the employee's time.

However, once a bug is discovered during the integration testing, the testing process is interrupted and the testers move on to other tasks. From the data provided by the Company, the majority of the bugs were discovered during the first two days of the testing. The testing is purposefully designed this way by prioritizing the test suits with the highest probability of finding a bug. That being said, the cost of the bug discovered during the integration testing on this project is estimated to be from 800 € to 1600 €. If we consider only the cost of the interrupted testing process, we can calculate the potential ROI for the integration testing as shown in table 3.4.3. As we can see, the number of bugs discovered during the integration testing phase to cover the initial investment depends on when the bug is discovered during the process. However, if we consider the average cost of the bug during this phase, it is sufficient to discover only 4 bugs to cover the initial investment for the project.

# bugs	Min(€)	% initial investment(€)
1	800	13% - 20%
2	1,600	26% - 40%
3	2,400	40% - 60%
4	3,200	53% - 80%
5	4,000	66% - 100%
# bugs	Average(€)	% initial investment(€)
1	1,200	20 - 30 %
2	2,400	40% - 60%
3	3,600	60% - 90%
4	4,800	80%-100% +
5	6,000	100% +
# bugs	Max(€)	% initial investment(€)
1	4,000	66% - 100%
2	8,000	100% +
3	12,000	100% +
4	16,000	100% +
5	20,000	100% +

Table 3.2: Potential ROI for integration testing - discovered bugs

The project manager must also account for the time developers will need to spend analyzing and fixing a discovered bug. This often causes a significant slowdown in the development process and can result in additional testing time. Experienced project managers have found that replicating and analyzing the root cause of a bug typically requires at least 4 hours of a skilled developer's time. The time required for actually fixing the bug varies depending on the specific issue but is generally consistent regardless of the phase in which it was discovered.

In summary, it takes 1-5 bugs discovered before the initiation of the integration testing by the VT framework to cover the initial investment. Besides the resources saved from the actual testing, there is additional time saved on the developer's side and the project manager's side. In addition, the development process is interrupted less frequently which ensures a more stable and predictable timeline for the project, which is one of the most important factors

for the stakeholders.

Bugs - System testing
The system testing is initiated after the integration testing is successfully finished. To better understand the expenses related to this testing phase, we need to consider following rules previously described in the section 3.1.1:

- Performed by the testers who are not involved in the integration testing
- Only physical hardware is used for this testing phase
- It is performed by a completely independent test framework that is purposefully unknown to the developers and the integration testers
- The project manager and the development team from the company have very limited control over the waiting periods and the time required for the testing
- In case of a discovered bug, the testing process needs to be repeated from the beginning including the integration testing
- The estimated time required for the system testing is 3-5 working days for 2-3 FTE(Full-time employee) testers

Considering the information above, the cost of one system testing round is estimated to be from **3,600 € to 6,000 €**, depending on the size and complexity of the project. For our hypothetical project, we are going to assume the cost of 2 FTE for 3-5 working days, which costs a fixed price of 50 € per hour. In addition to that, we need to consider the time required to hand over the project to the independent system testers and back to the developers. This price is estimated by the project manager to a fixed price of **800 €**.

Similarly to integration testing, system testing is purposefully designed to prioritize the test suits with the highest probability of finding a bug. For the purpose of this analysis, we will assume that the majority of the bugs are discovered during the first half of the testing as it has been observed in the past. The time for discovering a bug is estimated during the first twenty hours of testing, which costs up to **1,000 €**.

The details of the potential ROI for the hypothetical project are shown in the table 3.4.3. As we can see, even if we consider only the fixed price for handing over the project to the system testers, it is sufficient to discover only five bugs to cover the initial investment for the project. Needless to say, this situation is highly unlikely and it would typically take at least some time to discover the first bug.

It is important to note that prior to the initiation of the system testing after the bug fixing, the whole integration testing process needs to be repeated to make sure the changes did not influence other parts of the module. Therefore, the cost of the bug discovered during this testing phase is significantly higher as it also includes the cost of another round of integration testing(2400 € to 4000 €).

# bugs	Min(€)	% of the initial investment (€)
1	800	up to 20%
2	1600	up to 40%
3	2400	up to 60%
4	3200	up to 80%
5	4,000	66% - 100% +
# bugs	Average(€)	% of the initial investment (€)
1	1,800	up to 45%
2	3,600	up to 90%
3	7,200	100%+
4	8,600	100%+
5	10,400	100%+
# bugs	Max(€)	% of the initial investment (€)
1	6,800	100%+
2	13,600	100% +
3	20,400	100% +
4	27,200	100% +
5	34,000	100% +

Table 3.3: Potential ROI for system testing - discovered bugs

In summary, the potential ROI for the system testing is significantly higher than for the integration testing. Moreover, the VT framework has great potential to allow developers to verify the functionality of the module in various edge cases and scenarios that could be otherwise discovered at the end of the system testing, which costs the customer the most money and resources. In addition to that, the bugs discovered during the system, testing could easily result in a significant delay in the project's timeline or even a delay of other projects that the developers are assigned to.

Conclusion
The outcome of this analysis for the management and the stakeholders is clear. The VT framework has great potential to ensure a more stable and predictable timeline for the majority of the projects, and prevent unexpected expenses. Moreover, the initial investment for the customer is relatively low, and among other advantages, it takes up to 5 discovered bugs during the integration testing to cover them.

The potential return on investment for the system testing is even higher. Any bugs discovered during this late testing stage save the customer a significant amount of time and resources. It is essential to stress that the VT framework is designed to run particular test cases that would be otherwise discovered at the end of the system testing, where the potential expenses are the highest.

Besides the easily calculatable expenses, the VT framework prevents significant delays in the project's timeline that often cause the product to be released later than planned, which can lead to a loss of the customer's trust and a loss of potential revenue. However, both of these aspects are beyond the scope of

this analysis as it is difficult to estimate the potential and beyond the scope of the project manager's decision-making process.

This analysis is based on the information provided by the Company's management, project managers, developers, and testers. The stakeholders verified the numbers during the budget approval process during the project's initiation phase, and they were impressed by the potential financial benefits that a relatively small initial investment can achieve, which is even lower from their perspective as the adaptation of the framework could be started before the firmware development is initiated.

In conclusion, the initial investment for the VT framework is around 4,000 € to 6,000 € per project. For highly complex projects, the investment could be higher, but not significantly since the VT framework should not implement any additional logic and the it is designed to be easily adaptable to the project's requirements. The potential return on the investment is higher for complex projects with a higher probability of untested edge cases and scenarios. The project managers in the Company are advised to consider the adoption of the VT framework using a decision-making process similar to the one described in this analysis of the hypothetical project, which is based on the average and most common type of project in the Company. As the result of this analysis suggests, the VT framework seems to be a good investment for similar project types, and there is a high probability that the money invested in the VT framework will save significantly more on the testing process expenses.

3.4.4 Analysis summary

The analysis presented in this section provided a comprehensive overview of the potential impact of the VT framework on the project delivery process from a project management perspective. The analysis was divided into two three categories that the project managers consider: the impact on the project planning process, the impact on the Company's unified testing strategy, and the financial impact.

The analysis performed in subsection 3.4.1 showed that the VT framework is a great tool for developers and testers that also has a significant impact on the project planning process. We have identified that the most common source of unpredictability in the project planning process is the testing process which often requires additional time and resources after discovering a bug.

The VT framework addresses this issue by providing the developers with a way to create and run test cases from the integration testing test suite, and easily create their own test cases for various edge cases. Besides that, it adds a frequent automated feedback loop as a part of the CI/CD pipeline, as shown in figure 3.8. That improves the quality of the code and lowers the number of bugs discovered on the physical hardware, which lowers the risk of the necessity of additional testing rounds.

Another aspect that the management in the Company considers is how the new solution aligns with the unified testing strategy in the Company. The analysis shows that if the VT framework fulfills all of the requirements, it can be a valuable tool for testers and developers at all locations of the Company, despite the differences in their own testing frameworks. That brings various benefits such as reusability of test cases, easier bug replication, and unification of the way the tests are written by the developers and testers. Even though

this is not a direct financial benefit, it can significantly speed up the process of handing over the project from one team to another, which for many projects in the Company turned out to be a significant bottleneck that consumed a lot of time and resources.

The last aspect that was analyzed was the financial impact of the VT framework. The aim of the analysis was to determine how quickly the initial investment in the VT framework would pay off. That was done by identifying the estimated cost of bugs discovered during each phase of the testing process. The outcome of the analysis showed that the probability of positive ROI is high for the vast majority of the projects in the Company. However, the adoption of the VT framework still requires careful consideration by a project manager.

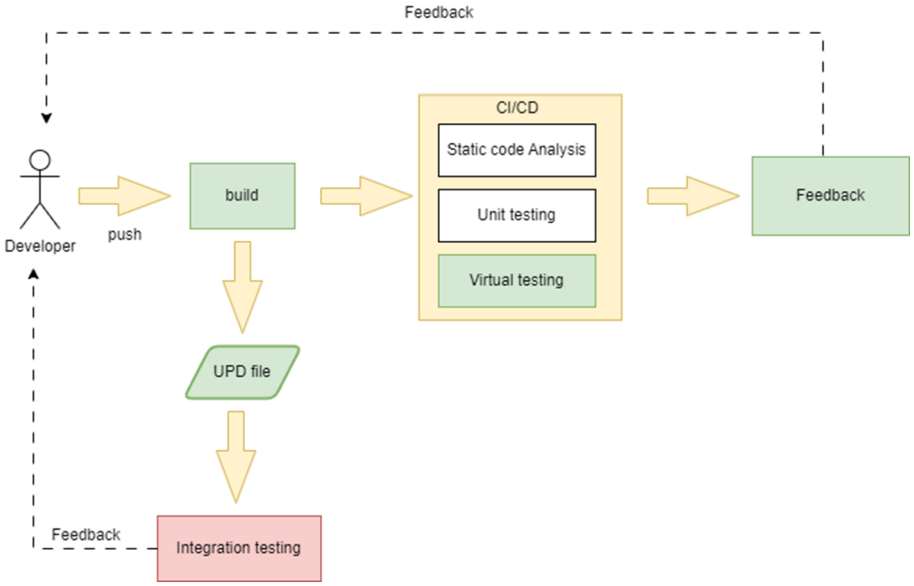

Figure 3.8: Automated testing in CI/CD pipeline - Feedback loop

The overall conclusion is that the VT framework can significantly improve the project planning process, reduce the risk of project delays, and have a positive financial impact on the project.

Chapter 4

Implementation

The aim of this chapter is to describe the implementation details of the VT framework based on the analysis and design performed in the previous chapter. The chapter is divided into several sections, each of which describes a specific component of the VT framework. As we are already familiar with the architecture of the framework, the focus of this chapter is on the implementation itself and the challenges that were faced during the implementation.

Before we dive into the details of the implementation, there are a few things that need to be mentioned. Since the Company is a large organization, it has its own standards and guidelines that need to be followed. One of the important rules that influenced the implementation of this project is to use Microsoft Visual Studio as the main development environment. For this reason, both the server application and the virtual testing library are implemented as a Visual Studio project. This decision could not be avoided and it was necessary to follow.

Another important thing to mention is the choice of the programming language. As all modules are being developed in C and C++, this programming language is always prioritized over others. One of the main reasons is it allows precise control over the memory and achieves great performance.

However, this does not apply to the testing clients where there are two main programming languages that testers use - Python and C#. Both of these languages offer great support for testing and are easy to use even for less experienced programmers. The performance limitations are not problematic here as the testing clients do not require any heavy computation.

4.1 Virtual testing library

The VT library consists of three main parts: the emulator, the DLL interface and the technology conde. Since the technology code is specific for each module, and cannot be published, a sample module was implemented to demonstrate the usage of the VT library. However, it is important to note that this code is not the actual technology code, but a simplified version of it.

For the reasons mentioned above, the VT library is implemented as a Visual Studio solution in C++. The project is divided into several directories as shown in the figure 4.1. The structure of the library is designed to be easily readable

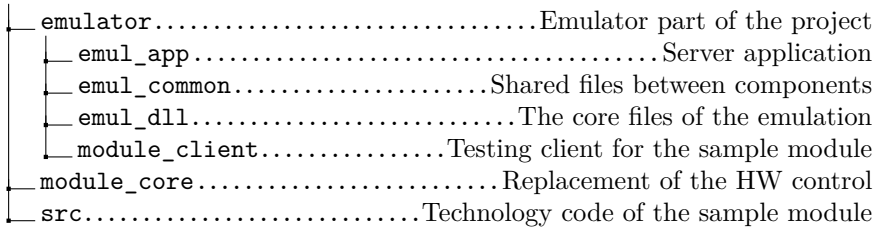

Figure 4.1: Directory structure - VT library

for developers and testers who decide to adapt the VT framework for their projects.

It is important to note that each directory can be assigned to a specific component of the VT library architecture. The data_obj directory along with any directory that includes the prefix "emul" is assigned to the Emulator component. The directories with the prefix "module" are assigned to the technology code or BaSy component, and the directories with the prefix "llc" are part of the DLL interface.

This clear structure corresponds to the requirements such as easy usage and maintainability of the framework. Another requirement that needs to be fulfilled is detailed logging of the emulator and the communication. For this reason, there is a separate directory called "log" that includes the logging functionality that is used throughout the whole library and the server application.

4.1.1 Emulator

The Emulator is the key component of the VT library. The most important files are located in the emul_core directory. The emulator is implemented as a set of functions that are used to call the appropriate technology code at the right time. In addition to that, it uses the appropriate data objects to store the data that is being exchanged between the technology code and the testing client.

Time emulation & Event handling
The I/O modules often call the technology functions at a specific time point, which makes it necessary to find a way to emulate the time in the VT library as well. However, since the VT framework does not strive to be a real-time testing system that would verify the actual behavior of the physical hardware, the time emulation could be simplified.

To ensure the right events are called at the appropriate time points, the emulator uses a mechanism called the "event calendar". The event calendar is a simple dynamic structure sorted by the time of the event. That allows all the components of the module to register their own events for the future. The emulator then iterates through the event calendar and calls the functions that correspond to the scheduled event.

The most common events are periodic events that are called at a specific time interval. That is why the event calendar allows each event to schedule its next call based on the current time point and the reload period.

To allow the event calendar to work properly, the events are represented by a class shown in figure 4.2. The Vector_Node class serves as a base class for all events that can be scheduled in the event calendar. If the event requires any additional variables or functions, the class can be inherited and extended, which is a common practice in the VT library.

```
class Vector_Node
{
private:
    uint64_t key;
    int TypeOfEvent;
    int NodeID;
public:
    void setKey(uint64_t timeStamp) {this->key = timeStamp;}
    void setTypeOfEvent(int eventType) { this->TypeOfEvent = eventType;
         }
    void setNodeID(int ID) { this->NodeID = ID;}
    const uint64_t getKey(){ return this->key;}
    const int getTypeOfEvent() { return TypeOfEvent;}
    const int getNodeID() { return NodeID; }
    void setVariables(uint64_t key, int TypeOfEvent, int nodeID);
};
```

Figure 4.2: Representation of an event in the event calendar

Data objects
Preparing required data objects is a crucial part of the adoption of the VT framework for a specific project. It is important to understand that the term "data object" is not used exactly as it is in the object-oriented programming context. The data object is any entity to which the VT library can assign a unique handle.

However, some data objects also require traditional object-oriented objects to store the data that would be typically stored somewhere in the hardware memory. An example of such a data object is the ds128 parametrization structure that holds the currently applied parametrization of the I/O module.

The handles are communicated between the testing client and the VT library during the initialization of the communication. Without the handles, the testing client cannot access the data objects and therefore cannot control the module.

Emulation
The execution of the emulation is always triggered by the testing client. Once the client sends the command to start the emulation, the emulator starts to iterate through the event calendar and call the corresponding technology functions. The highest priority always has the event that is scheduled for the closest time point. Once the event is processed and executed, the event is removed from the event calendar if it is not a periodic event, which only reschedules the next call.

The emulation always runs for a given number of milliseconds of the artificial time, which is necessary to allow the testing client to check the internal state of the module by reading the corresponding data objects. In other words, the

internal state of the module is updated only when the emulation is specifically requested by the tester.

The figure 4.1.1 shows the implementation of the execute function that is called through the DLL interface. The argument of the function is the number of milliseconds that the emulation should run. The goal of this function is to process all events that are scheduled between the current time point and the given time point. The function works as follows:

1. The function calculates the end of the iteration based on the current time point and the given time point.

2. The function processes all events that are scheduled before the end of the iteration.

3. The function resets the current time point and the breakpoint to zero to prepare for the next iteration.

4. The function returns the remaining time that was not processed, due to the lack of scheduled events in the time frame.

To summarize how the emulation works, the testing client sends the command to start the emulation for a given time period. It is crucial to understand that the emulation does not run in real-time, but in an artificial time that is controlled by the event calendar. The time with no scheduled events is skipped to the first event that is scheduled in the future. The testing client can then read the data objects to check the internal state of the module before and after the emulation. The tester needs to be aware that without the execution command, the technology code is not called and the internal state of the module is not updated despite setting the parametrization or any other data object.

```
uint64_t emul_execute_emulation(uint64_t ms) {
  uint64_t ret = ms;
  BreakPoint += ms;
  new_emulation_process = true;
  EndOfIteration = CurrentTime + BreakPoint;
  Process_events();

  for (int i = 0; i < Event_index; i++) {
    if (Events_Array[i]->getKey() >= EndOfIteration) {
      ret = Events_Array[i]->getKey() - EndOfIteration;
      Events_Array[i]->setKey(Events_Array[i]->getKey() - (
          EndOfIteration));
    }
  }

  CurrentTime = 0;
  BreakPoint = 0;
  module_report();
  return ret;
}
```

Figure 4.3: Implementation of the execute function

4.1.2 DLL interface

The DLL interface defines how the emulator functions are called from the testing client. This component is module-specific and must be adjusted for each module that is being tested. However, the set of functions that the DLL interface should provide stays the same with very few exceptions.

The interface includes the following functions:

- **start_and_initialize:** Responsible for processing the initialization command.
- **query_do_handle:** This function is used to get a handle on any data object.
- **write_data_record:** This function is used to write a data record that parametrizes the module.
- **read_data_record:** Returns the data record that is currently applied to the module.
- **write_do_data:** Write the data to the data object based on the given handle.
- **read_do_data:** Read the data from the data object based on the given handle.
- **execute_emulation_ns:** Execute the emulation for a given number of nanoseconds.
- **insert_submodule:** The majority of modules are multiple variants called submodules. The submodules usually differ in their input/output data length or the size of the data records. To allow the technology code to verify the correct size of the data, the submodule must be inserted.

These eight functions are the core of the DLL interface that allows to fully control the module and verify its behavior from the testing client.

4.1.3 LLC communication layer

All communication between the testing client and the VT library is done through the Low-Level Commands. The LLC layer is responsible for following the pre-defined protocol shown in figure 3.6. The implementation of the LLC layer consists of the following functions:

- **dll_llc_payload_start:** Initiates parsing/composing of parameters.
- **dll_llc_payload_rx_finish:** Completes processing of the request.
- **dll_llc_payload_tx_finish:** Completes composing of the answer.
- **dll_llc_payload_error:** Concludes interface function with an error, taking an error code as input.
- **dll_llc_payload_get_uint32:** Retrieves a UINT32 from the input frame.

- **dll_llc_payload_put_uint32:** Inserts a UINT32 into the output frame.
- **dll_llc_payload_get_uint16:** Retrieves a UINT16 from the input frame.
- **dll_llc_payload_put_uint16:** Inserts a UINT16 into the output frame.
- **dll_llc_payload_put_uint8:** Inserts a UINT8 into the output frame.
- **dll_llc_payload_get_name:** Retrieves an IDENTIFIER from the input frame, with a maximum length specified.
- **dll_llc_payload_get_data:** Retrieves actual parameters, with the length specified.
- **dll_llc_payload_put_data:** Inserts a data array into the output frame, with the length specified.

These functions are used to parse, compose, and send the LLC commands between the testing client and the VT library. The data are being exchanged in the form of frames, where each frame has a header that specifies necessary information about the command and the payload that contains the actual data.

4.1.4 Sample module

The sample module is a simplified version of the technology code that is used to demonstrate the usage of the VT library. The sample module does not have any real functionality, however, it can be parametrized in many ways to demonstrate the typical behavior and the usage of the VT library.
The module is represented by a class that implements the following functions:

- **Constructor:** Initializes module data with default values.
- **report_module:** Reports module data.
- **module_init:** Initializes the module.
- **module_open:** Opens the module.
- **module_start:** Starts the module.
- **set_default_module_ID:** Sets the default module ID.
- **set_submodule_ID:** Sets the submodule ID.
- **configure_module:** Configures the module with a new channel and mode.
- **perform_services:** Modifies module's variables according to its algorithms.
- **parametrize_module:** Parametrizes the module based on provided data.

- **is_module_initialized:** Checks if the module is initialized.
- **is_module_opened:** Checks if the module is opened.
- **is_module_started:** Checks if the module is started.
- **get_module_ID:** Retrieves the module ID.
- **get_channel:** Retrieves the channel value.
- **get_mode:** Retrieves the mode value.
- **get_sub_mod_id:** Retrieves the submodule ID.
- **get_WB_Enabled:** Retrieves the status of WB diagnostics for a specific channel.
- **get_SC_Enabled:** Retrieves the status of SC diagnostics for a specific channel.

The internal state of the module is stored in the class variables that are being updated through the technology functions. That allows the testing client to read the internal state, verify its correctness, and check the behavior in the same way as it would be done with the real technology code. The only difference is that instead of complex algorithms, the sample module is set to return a predefined value that is based on the information that the tester provides through macros.

The sample module currently provides the following functionalities:

- **Initialization:** The module must be initialized before it can be used. The initialization is successful or unsuccessful based on the macro that is set by the tester.
- **Parametrization:** The module can be parametrized with a data record that is provided by the testing client. However, the parametrization is simplified to the more common use cases such as diagnostics settings.
- **Configuration:** The module can be configured with a new channel and mode. The configuration can be even unsuccessful based on the macro that is set by the tester.
- **Diagnostics:** The module allows to enabling diagnostics of WB (Wire break) and SC (Short circuit) for a specific channel.
- **Input/Output control:** The module allows setting the input values as well as reading the output values.

In addition to the technology functions, a functioning module does need a layer that controls the module's hardware. The hardware layer is mocked by the files in the "module_core" directory. That corresponds to the typical architecture of the I/O modules described in the previous chapter.

These functions do not control the real hardware, however, they simulate the same behavior for the technology code that is being tested and is stored in the "src" directory.

In summary, the sample module is a simplified version of a real I/O module that can be used to demonstrate the usage of the VT library and the whole VT framework. The module is designed to be easily parametrized to allow us to test different scenarios and verify the functionality of the already implemented components.

4.2 Server application

The server application is a component of the VT framework responsible for the communication between the testing client and the VT library. As we can see in the figure 4.4, the server application handles the commands from the client through the functions provided by the DLL interface.

Similarly to other components of the VT framework, the emul_app is implemented as a Visual Studio project in C++. Since the Microsoft Windows operating system is mandatory to use in the Company, there is currently no requirement to support other operating systems.
The application works in the following steps:

1. The application is started and waits for the connection from the testing client.
2. Once the client is connected, the application loads the DLL testing library.
3. The application processes the commands from the client and handles the inter-process communication
4. Once the client disconnects, the application unloads the DLL library and waits for the next connection.

During the implementation, the most important issue that needed to be resolved was identifying the most appropriate communication channel. There were different options available, such as packets or pipelines. However, since the solution is already reliant on the Windows operating system, selecting a Windows pipeline was a viable choice. This option offers a highly efficient communication method between processes, while also providing error-handling capabilities. It is also important to mention the name of the pipeline. Therefore, the clients need to be aware of the name that is supposed to be used. In case the name of the pipeline needs to be changed on the server side, the application offers a command line argument that sets the pipeline's name to any valid string.

Besides the command line argument mentioned above, the server application offers the following list of other arguments that allow the user to configure the server as required:

- **location <path_to_dll_file>:** Specifies the path to the DLL file. Example: `-L ..\example.dll`
- **timeout <timeout>:** Sets the IPC timeout in seconds after which the client is disconnected. Example: `-T 10` or `-T 0` for infinity.
- **infinite:** Enables infinite run of the emulator.

- **pipe-name:** Sets the name of the pipe. Example: `-P my_pipe`
- **log-level <lvl>:** Sets the log level.
- **colored:** Enables colored terminal output.

The command line arguments above provide the tester with complete control over the server. As the application is straightforward to restart and quite simple, there was no need to modify the server settings while running. It is important to note that the log-level setting applies not only to the server application but also to the VT library.

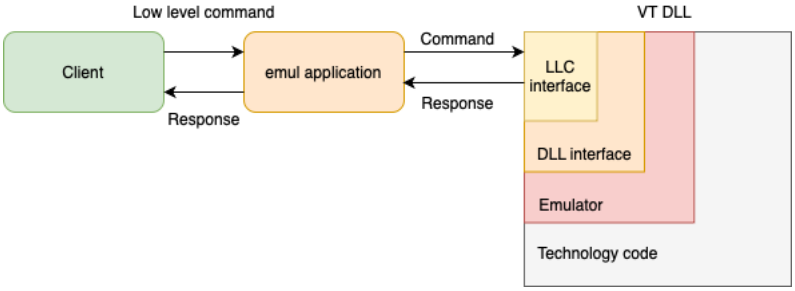

Figure 4.4: Communication layers

4.3 Testing client

As we know from the architecture of the VT framework, the testing client is any application that implements the pre-defined interface for the VT library. Similar to the sample module, this section describes a simple implementation of the testing client. To conveniently show the functionality of the VT framework, the following testing client is designed to test the sample module described in the section 4.1.4

The client can be divided into three layers:

- **emul_llc layer:** This layer is responsible for communication with the server using the Low-Level Commands.
- **tech_control:** A layer is responsible for the control of the technology code such as parametrization, initialization and input/output control.
- **Test cases:** The client includes a set of test cases that uses the tech_control layer to test the sample module.

It is worth mentioning that the emul_llc layer is a common part of all testing clients implemented in Python. There is no need to adjust this layer for each module that is being tested as the communication protocol does not change and the layer is only responsible for data exchange not validating the data.

On the other hand, the tech control layer is usually specific to the module and requires adjustments at the start of the testing client implementation. Nevertheless, most functions have common headers, and only the implementation

of the functions needs to be customized. Therefore, this section emphasizes the general functionality of these functions rather than their implementation details. The most commonly recommended functions to be implemented in the tech control layer are:

- **query_do_handle:** This allows the client to get a handle on the data objects that are used to control the module.
- **set_submodule:** This function is used to set the submodule ID.
- **set_channel:** This function sets the module input value at the specific channel.
- **get_input_data:** This function reads the input data from the module.
- **get_channel_di:** This function reads the DI (Digital Input) value from the module.
- **get_channel_qi:** This function reads the QI (Quality Information) value from the module.
- **get_diagnostics:** This function allows the client to read the diagnostics status of the module.

The majority of test cases are implemented using the functions above, in case the tester needs to test a specific functionality that is not covered by these functions, the tech control layer can be easily extended with new functions.

Similar to the data objects in the VT library, it is a good practice to use corresponding data objects in the testing client. These data objects are used to store the data in a predefined format and prevent the tester from making mistakes when setting the data.

A good example of such a data object is the parametrization structure that is used to parametrize each channel of the module. The parametrization structure for I/O modules is usually quite complex and the majority of information is stored in two to three bytes of data, which is not easily readable for the tester.

The last layer of the testing client is the set of test cases. As we know from the previous chapters, the test cases should not differ from the test cases that are used in the real testing environment, which is why the test cases need to use the tech control layer so all of the required changes can be done in the implementation of the tech control layer and not the test cases themselves.
All test cases should follow the following steps:

1. **Initiate module start and initialization:** During these steps, the client should check the connection to the server application and request the required handles on the data objects. Then send a command to initialize the module.
2. **Parametrize the module through the parametrization structure.** Each test case should parametrize the module at the beginning. Without a proper parametrization, the behavior of the module is undefined and therefore it does not make sense to verify it. Besides that, the module can be reparametrized during the run and even parametrized with multiple parametrization structures.

3. **Set input values & verify output values:** The majority of test cases for I/O modules are based on setting the input values and verifying the output values which should be based on the input values. It is crucial to execute the emulation after setting the input values to allow the technology code to process the input values and set the output values.

4. **Verify the diagnostics status:** The diagnostics status is an important part of the I/O modules. Even if the output values are correct, that does not mean the module is working correctly. The technology code has many mechanisms to detect these errors and the diagnostics status is the way to verify them.

The usage of the testing client is shown in the chapter 5 where the client is used to test the sample module. It is crucial to understand that this is only one of the possible implementations of the testing client and the client can be implemented in any the programming language that implements the DLL interface.

4.4 Logger

Logging is a crucial requirement given by the developers and testers as it was mentioned in the chapter 3.2.1. A good logging system is necessary to understand the behavior of all of the components and to be able to debug the system in case of any issues.

The potential issue with the logging system is choosing the right level of logging and to provide the user with the possibility to change the level of logging. Without this feature, the log files can grow to a size that is not manageable and the important information can be lost in the noise.

For this reason, the logging system offers the following levels of logging:

- **Debug:** Provides detailed information for debugging purposes.
- **Info:** Provides general information about the application's state.
- **Warning:** Indicates potential issues that may need attention.
- **Error:** Indicates critical errors that require immediate attention.

The only way to set the log level for the server application is by using the command line argument. Once set, the log level is applied throughout the entire application, including the VT library. The log level is assigned to each log message as shown below this paragraph, and the message is only written to the log if the log level of the message corresponds to the log level set by the user. The only exception is the DEBUG log level, which serves as an extension of the INFO log level and usually provides more detailed information about the current state of the server application or the VT library.

```
Logger::log(LogType::Info, message);
```

Another challenge that needed to be faced during the implementation was the common logging system for the server application and the VT library. Since these two components are closely related, it would have been confusing to have

two separate logs. However, the server application is a standalone application that needs to be able to run without the VT library.

To tackle this issue, an independent message queue that runs in a separate thread was created. The message queue stores the log messages sent from the components of the VT library. The server application then reads the message queue, and the log messages are displayed in the console.

To use the logger in the VT library component, the logger class needs to be included in the component's header file. Once included, an instance of the message queue needs to be created, as shown in figure 4.5. It is crucial to use the namespace Logger to avoid any conflicts with other libraries that may use the same name for the logger class.

To sum it up, the logger fulfills the requirements of the developers and testers and provides a detailed log of all the components of the VT framework. The usage of the logger is straightforward and does not require any additional knowledge, besides the log levels that are used to filter the log messages and creating an instance of the message queue for your component.

```
namespace Logger {

extern MessageQueue msgQueue;

template <typename T>
requires emul_concepts::printable<T>
void log(LogType::type logLvl, T message) {
    msgQueue.push(logLvl, message);
}

extern MessageQueue* getMsgQueue();
}
```

Figure 4.5: Example of the logger usage

Chapter 5

Demonstration of the VT Framework usage

The chapter aims to demonstrate the functionality of the Virtual Testing Framework which corresponds to point 4 of the objectives of this thesis. Since it is not possible to publish an existing module, the demonstration will be shown on a sample module described in section 4.1.4. The module has very simple functionality which can be easily parametrized through the macros in the header file. That is convenient for the demonstration purposes as we can easily change the behavior of the module.

Besides showing the functionality of the VT framework, this chapter should also serve as a simple user manual for the potential users of the VT Framework, including the installation, prerequisites, test case preparation, and execution of the test cases.

5.1 Prerequisites & Installation

All parts of the VT Framework are developed as a Visual Studio solution utilizing the platform toolset of Visual Studio 2022 (v143). To ensure optimal functionality, it is recommended to use the same version of this Microsoft IDE. Once installed, no further installation is necessary, and the solution can be compiled and executed with ease.

Keep in mind that the emul_dll project is set to start the server application through a command line argument. Therefore it is necessary to compile and build the emul_app project first. Without the emul_app exe file, the emul_dll project will not be able to start the server application and the testing will fail.

The last step required to test the sample module is to run the testing client. The testing client is implemented in Python, therefore it is necessary to have Python installed on the system. Once the Python is installed, the testing client can be run directly from the command line or in the Visual Studio IDE.

5.2 Test case preparation

One of the main features of the VT Framework is the ability to run the same test cases on the physical hardware and the virtual environment. To achieve

this goal, the test cases must use the functions from the tech_control layer which can differ in the implementation based on the environment.

To create a new test case, the user must follow the steps below:

1. Create a new Python file in the testing client directory
2. Import the necessary modules from the tech_control layer such as the input/output codes, parametrization structures or and functions for controlling the module
3. Create an instance of the emulation library
4. Use the created instance to call the functions from the tech_control layer

The testing client is designed to handle the communication outside of the scope of the test cases. Therefore the user does not need to worry about the LLC protocol or inter-process communication between the client and the server application.

Once the formal part of the test case is prepared, the user can implement the test case logic. The tests for the I/O modules usually consist of the following steps:

1. Module initialization and parametrization
2. Assert the initial state of the module
3. A loop of setting the input values and checking the validity of the internal state of the module including its output values
4. Report the results of the test case

The testing client directory contains multiple examples of the test cases for the sample module. Part of the test case named "test_square_signal.py" is shown in figure 5.1 and will help us to demonstrate how the test case should be implemented.

The test case is designed to test square signal generation on the sample module. As you can see, the test case repeatedly sets the input value to low and high and checks the output value and the quality information bit. Since the sample module is not supposed to change the signal in any way, the output value should be the same as the input value after the execution of the module. If the assert statement fails, the test case will stop and the user will be informed about the failure.

```
# Initialization and parametrization of the module


for i in range (0, 10):
    # Set the input value to low
    el.set_channel(0, Input.SAMPLE_MODULE_LOW)

    # Execute emulation for 70 ms
    el.wait_ms(70)

    # Check the output value and quality bit
    assert el.get_channel_di(0) == 0
    assert el.get_channel_qi(0) == 1

    # Set the input value to high
    el.set_channel(0, Input.SAMPLE_MODULE_HIGH)
    # Execute emulation for 70 ms
    el.wait_ms(70)

    # Check the output value and quality bit
    assert el.get_channel_di(0) == 1
    assert el.get_channel_qi(0) == 1
```

Figure 5.1: Example of the test case for the sample module

5.3 Module parametrization

Every test case must parametrize the module at the beginning of the test case. The parameterization has a crucial impact on the behavior of the module and it does not make sense to test its functionality without knowing what to expect from the module.

The parameterization is done through the parametrization structure which is defined in the technology control layer.

5.4 Diagnostics testing

Besides testing the input and output values of the module, the VT Framework also allows the user to test the diagnostics information that the module provides. The diagnostics information is usually used to find out what went wrong with the module in case the quality information bit is set to 0 for the specific channel.

It is important to note that the diagnostics are not enabled by default. The module must be properly parametrized to provide the diagnostics information. In addition to that, there are multiple types of diagnostics such as: WB (Wirebreak) diagnostics or SC (Short Circuit) diagnostics.

In the attachment of this thesis, you can find two test cases for the diagnostics testing. The first test case named “test_WB.py” is designed to test the Wirebreak diagnostics as shown in figure 5.2. The point of the test case is to test if the diagnostics information is available only if the diagnostics are enabled for the specific channel. As you can see, the diagnostics are enabled only for the first channel and the test case tries to set the input value to the first

and second channels. The desired behavior is that the diagnostics information is available only for the first channel.

For a better understanding of this test case, the figure 5.3 hows the flow of the test case. As we can see the test case starts with establishing the connection with the server application. Then the module needs to be properly parametrized to enable the diagnostics on the desired channels. After that, we perform the test steps that verify the behavior of the module by checking the output value, quality information bit, and diagnostics information. If the assert statement fails, the test case will stop and the user will be informed about the failure.

The second test case named "test_SC.py" is designed to test the Short Circuit diagnostics. The test case is almost identical to the previous one and any other test case that verifies the behavior of the diagnostics. Typically, the only difference is in the type of the diagnostic.

In summary, to show the functionality of the VT Framework for the diagnostics testing, two test cases were implemented for the sample module. The test cases can be found in the attachment of this thesis in the "sample_module_client" directory and can be run directly from the command line or in the Visual Studio IDE as described in the user manual in Apendix B.

```
# Enable the WB diagnostics for the first channel
ParStructure.ch[0].diag_wb = True
ParStructure.ch[1].diag_wb = False

el.power_on()

# Parametrization
el.set_submodule(ModuleID.SAMPLE_MODULE_V2, SubmoduleID.
    DEFAULT_SUBMODULE)
el.write_data_rec(128, bytearray(ParStructure))
ret, _ , data_record = el.read_data_rec(128, len(bytearray(
    ParStructure)))
assert data_record == bytearray(ParStructure)

for i in range (0, 10):
    # Set WB for the first channel
    el.set_channel(0, Input.SAMPLE_MODULE_WB)
    el.wait_ms(100)
    assert el.get_channel_di(1) == 0
    assert el.get_channel_qi(1) == 0
    assert el.get_diagnostics(1) == DiagOutput.DIAG_WB


    el.set_channel(1, Input.SAMPLE_MODULE_WB)
    el.wait_ms(100)
    assert el.get_channel_di(1) == 0
    assert el.get_channel_qi(1) == 0
    assert el.get_diagnostics(1) == DiagOutput.DIAG_NO_ERROR
```

Figure 5.2: Simplified example of the test case for the sample module

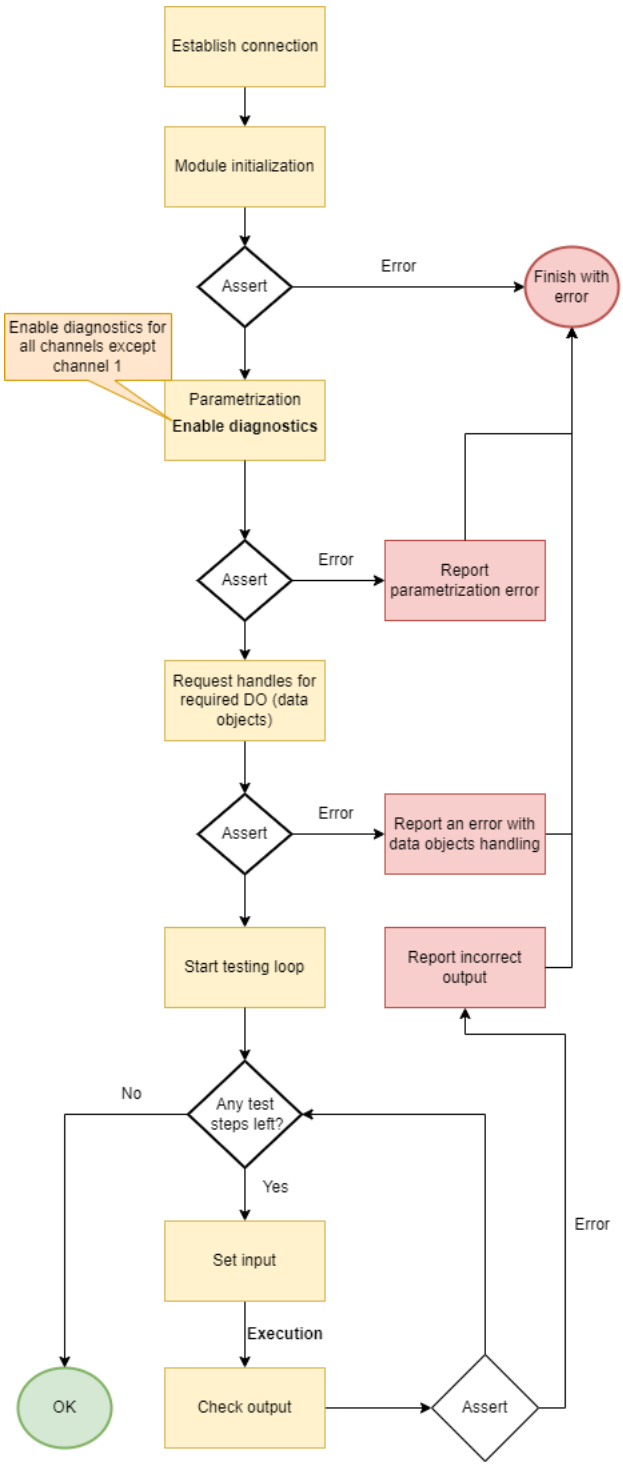

Figure 5.3: The flow of the Test_WB Test case

5.5 Logging & Reporting

The VT Framework has a crucial requirement of providing the user with a comprehensive logging system. This will help the user to have a clear understanding of the internal state of the module at any given time and to easily find out what went wrong in case of a failure. Additionally, the framework should also generate a report that summarizes the results of the test cases to allow the testers to easily recognize if a bug was identified or not.

This section will demonstrate the logging and reporting functionality of the VT Framework of the VT library, the server application, and the testing client. However, since the testing client should be easily replaceable by the user's own testing client, the demonstration will be focused on the components of the VT library and the server application.

In the previous chapters, we have already discussed the complex logging system that is shared between the VT library and the server application. This logging system allows the user to set the logging level to the following values: DEBUG, INFO, WARNING, ERROR. The logging level can be set through a command line argument of the server application as shown in figure 5.4.

```
-L ..\..\emul_dll\Debug\emul_dll.dll -I --log-level INFO
```

Figure 5.4: Command line argument for setting the logging level

The logging level of each individual message is set by the developer of the technology control layer. It is a good practice to use the ERROR level only for messages that indicate a severe problem during testing. The example of the usage of the logging system is shown in the figure 5.5.

```
// get name
if (!dll_llc_payload_get_name(&data_obj_name, sizeof(rx_frame->
    query_do.do_name))) {
    Logger::log(LogType::Error, std::format("{} : Get name- ERR
        ", __func__));
    return;
};
// finish the request
if (!dll_llc_payload_rx_finish()) {
    Logger::log(LogType::Error, std::format("{} : Finish
        payload - ERR", __func__));
    return;
};
```

Figure 5.5: Logging macro for the sample module

As we have introduced the reader to how to use the logging system in the VT library and the server application, we can now show the output of the logging. The output is displayed in the console window and can be easily redirected to a file. As the figure 5.6 shows, the has a specific format that includes the timestamp, the logging level, and the message itself. The example shows a logging output of the "write_data_record" function that parametrizes the module. As you can see, the INFO level is used for general information

about the function, while the DEBUG level is used to inform the user about the details of the data record and other values that are important for this particular function. Keep in mind that none of these messages would be displayed if the logging level was set to ERROR or WARNING.

```
[2024-03-28 22:41:38 -  INFO  ] EMULDLL_iface_write_data_record
[2024-03-28 22:41:38 -  DEBUG ] EMULDLL_iface_write_data_record: record
     number: 128, record len 50
[2024-03-28 22:41:38 -  DEBUG ] EMULDLL_iface_write_data_record: Check
    para ds - success
[2024-03-28 22:41:38 -  DEBUG ] Reading of rx_frame - done; Record
    number: 128
[2024-03-28 22:41:38 -  DEBUG ] Curr data record len: 0; Data object
    size : 50
```

Figure 5.6: Logging output of the VT library and the server application

Besides this logging, the VT library also provides various logging macros which are parametrized by static variables in the header file. These macros are convenient if the user wants just quickly log some information about the module without the need to use the complex logging system. An example of the logging macro for the sample module is shown in figure 5.7. The macro is defined in the header file of the sample module and can be used in the source file to log the information about the inner state of the module.

```
#define LOG_SAMPLE_MODULE(flag,...)      \
    do {                                 \
        if (flag) {                      \
                printf("MODULE --> ");       \
                printf(__VA_ARGS__ );        \
            };                               \
        } while(0)
```

Figure 5.7: Logging macro for the sample module

The output of this particular logging macro would be also displayed in the console window and looked as shown in the figure 5.8.

```
MODULE --> ----------------------
MODULE --> Initialized: true
MODULE --> Opened: true
MODULE --> Started: true
MODULE --> Parametrized: true
MODULE --> Module ID: 1
MODULE --> Submodule ID: 264
MODULE --> Channel: 0
MODULE --> Mode: 1
MODULE --> Services Performed: 12160
```

Figure 5.8: Output of the logging macro for the sample module

In summary, the VT Library offers a complex logging system that is capable of gathering information about the current state from all the individual

components of the VT Library, including the server application. The actual reporting of the test results is done by the testing client because the VT Library is not aware of the test cases that are being executed, nor the expected outputs.

Chapter 6

Evaluation of the Virtual testing framework

The VT framework project has been implemented in multiple projects at the Company. Therefore, it was possible to collect valuable feedback and evaluate the impact on the projects and the user experience. This chapter aims to summarize the data collected from the users of the VT framework and evaluate whether the project goals and requirements were fulfilled or not. Once the solution is evaluated, the focus of this chapter will shift to the possible improvements that could be implemented to improve the framework's usability.

6.1 Requirements fullfillment

This section aims to evaluate the provided feedback and determine if the requirements given in the section 3.2.4 were fulfilled. The requirements were divided into three categories: usability, quality of testing, and resource management.

Usability
The usability category primarily focuses on the VT framework adoption on new projects. Therefore, the following requirements are included:

- **Easily applicable:** How hard was it to adapt the VT framework for your project?
- **Compatibility with solutions across all Company locations:** Is the VT framework compatible with various testing clients?

The most significant requirement from the usability point of view is the complexity of VT framework adoption for new projects. According to the analysis performed in section 3.4.3, it takes two to three weeks of work of one experienced developer to adapt the VT framework. The provided feedback suggests that both teams were able to succeed in this task in the given time frame. The predictability of the initial cost is crucial for the project managers, especially for shorter projects where the positive ROI can be questionable. Therefore, fulfilling this requirement is considered one of the key indicators for this project.

The compatibility requirements were tested on two testing clients as a proof-of-concept at two most important locations of the Company as far as the I/O module development is concerned. In both cases, the required modifications at the testing client-side were implemented within three weeks. The interface was described as clear and easily implemented by the experienced testers. Whether the VT framework will be used at another location remains unknown.

In summary, the provided feedback suggests that the VT framework is easily applicable to the new projects, including the required implementation of the communication interface.

Quality of testing

One of the goals of this project is to improve the quality of testing by increasing the frequency, improving the module control, and providing a type of gray box testing for the testers and developers.

The most important requirement that increases the frequency of testing is the integration into the CI/CD pipeline. Every time a developer adds code to the project repository, regression testing should be performed to confirm the newly added code does not negatively influence any already existing features. Typically, each team services their own testing pipeline according to their needs. However, the general approach is to divide the automated jobs into steps to make the pipeline easily readable. Therefore, the usage of the VT framework pipeline was divided into the following steps:

1. Build the VT library project, which includes the technology code of the testing module
2. Publish the VT library as an artifact accessible for usage
3. Build the server application
4. Create a virtual environment for the virtual testing client
5. Install all required packages for the testing client
6. Run all test cases from the dedicated directory
7. Publish the results

The provided feedback suggests that the following steps were easy to follow and the integration to the CI/CD pipeline took approximately an hour of work. Another important aspect of this feature is the time required to run the tests. To provide an accurate evaluation, the teams were asked to transfer all of the existing test cases and evaluate how long it takes to get the results.

The results suggest that for both projects, around 80% of the test cases from the existing test plan were transferable to the VT framework. The remaining 20% were not transferable due to their high dependency on the physical hardware e.g., LED testing. However, this situation was fully anticipated and 80% of the test cases are considered a success. In both cases, it took less than twenty minutes to run all of the transferable test cases from the test plan. That means, once the developer pushes a new code to the repository, it takes less than twenty minutes to get the results from Static code analysis, Unit tests, and Virtual tests.

In addition to testing in the CD/CD pipeline, testing frequency can be increased by allowing developers to run tests on their local machines. This feature is closely related to the need for gray box testing and improving test coverage by enhancing module control. The primary concern of project managers regarding this approach is that implementing test cases may be too time-consuming for developers since they have very little knowledge about creating test cases. For this reason, the project managers were asked to make a comprehensive evaluation using the VT framework on one of the projects.

Therefore a senior developer who did not participate in the VT framework adaption for the project was asked to implement a test suite that consists of twenty-one test cases. This time was then compared to the time that an experienced tester needed to implement the same test suite. The result of this experiment shows that the developer needed approximately 8-12 hours to understand how the VT framework works and how the test cases should be implemented. After this initial phase, the average time needed to implement one test case was around 24 minutes. However, it is worth mentioning that with each implemented test case, the time needed for the implementation decreased as figure 6.1 shows. The average time required for the first 4 test cases was over 38 minutes, while the average time for the last 7 test cases was around 20 minutes. This decrease in the required time was an important factor for the project managers, as it suggests that the developers can quickly learn how to implement test cases using the VT framework.

The experienced tester did not need the initial 8-12 hours to understand the VT framework, as he was already familiar with it. The average time needed to implement one test case was around 20 minutes. This result was surprising for the project managers as it suggests that the developers can quickly learn to implement test cases using the VT framework at a similar pace as the testers.

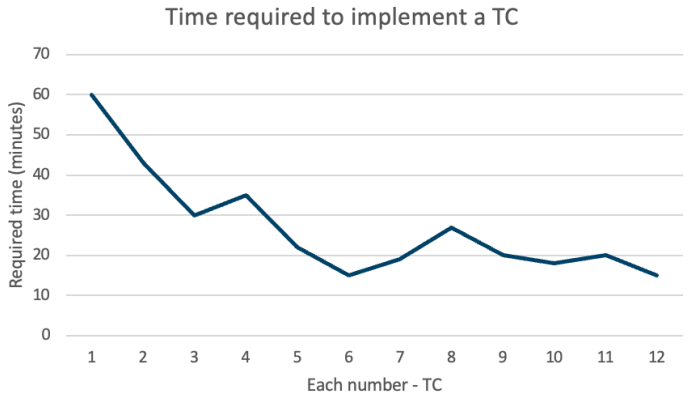

Figure 6.1: Time needed to implement one test case

The results of this experiment suggest that the VT framework is easy to use and allows developers to create their own test cases relatively quickly. Besides that, most of the test cases the developers create can be reused on the physical hardware, which increases the test coverage without any additional effort from the testers. In addition, the VT framework allows the implementation of test

cases with exact control over the module, which is impossible to achieve on the physical hardware.

Gray box testing
Another requirement that is closely connected to the quality of testing and improving the module control, in general, is to fill the gap between the black box and white box testing that the Company is currently using. The goal is to allow the user to execute the test cases with precise control over the module and the environment, being able to stop the execution at any given time and check the state of the module.

This requirement was successfully fulfilled by a combination of the implementation of the event calendar and step-by-step execution described in section 4.1.1, which allows the execution of the module for any given time beginning with one nanosecond. Once the execution is over, the testing client can request the data from the module and verify if it is correct.

In case the data stored in the data objects are not enough to verify the internal state of the module. The behavior of the whole VT library and the server application can be debugged by the Visual Studio debugger. That provides an option to set all types of breakpoints inside of the technology code and observe the code execution line by line, which is nearly impossible to achieve on the physical hardware. This feature turned out to be highly appreciated by the developers during the development as they are all familiar with the Visual Studio debugger and it was very difficult for them to verify the newly implemented technology code on their local machines.

6.2 Resourse management

The goal of the VT framework is to lower the necessary resources for testing by resolving some of the inefficiencies of the testing process. The expected financial impact is described in section 3.4.3. The analysis describes the anticipated initial cost and how many bugs need to be discovered by the VT framework to achieve a positive financial impact. The analysis is focused on three main topics: the initial investment, the impact of the VT framework on the speed and predictability of the development and the impact of the discovered bugs.

The first topic is to evaluate the initial investment required to adapt the VT framework for the project. This investment is anticipated to be the work of one experienced developer for two to three weeks. The given time frame was confirmed in both cases and it is reasonable to assume that the initial investment is predictable and within the acceptable range.

Secondly, according to the developer's feedback, the impact on the speed of the development is positive. Especially at the beginning of the project when it typically takes more than a month to obtain the required hardware and start the development. In both cases, the waiting period for the hardware exceeded four weeks. During this period the developers are assigned to the project, unable to work on the actual development and therefore working on the low-priority tasks.

Once the VT framework was adapted to the project, the developers could immediately start with the development. That means the actual development started two to three weeks earlier than it would without the Virtual testing.

Besides that, it allowed the integration testers to start working on the basic test cases described in the test plan. That means once the required hardware arrives, the basic features of the module are already implemented and can be only verified on the physical hardware. The cost of two weeks of development varies depending on the complexity of the project. However, even for small projects with three developers, the cost exceeds 18,000 €, which covers the initial investment to the VT framework by itself.

In addition to the saved time and resources at the beginning of the project, the developers highly appreciate the possibility of easily executing and debugging the newly implemented code. Until now the developers were required to use complicated racks with the physical hardware to help them find the root cause of many complex bugs. According to the feedback, the VT framework helps them to resolve the bugs faster and with less effort as they can easily identify the problematic section and debug it line by line.

Lastly, the project managers need to consider the financial impact of the bugs discovered early in the development by the virtual testing. There is no direct way to measure the number of bugs that would otherwise be discovered later in the testing process. However, according to the project managers of the projects, no test cases that passed in the virtual testing environment failed on the physical hardware. That suggests that the testing is quite reliable and provides reliable feedback to the developers and testers which undoubtedly will have a positive impact on the project budget.

In summary, after the evaluation performed by the project managers based on the feedback from the developers and testers, the VT framework is considered a successful project from the quality of testing and resource management point of view. The goal of lowering the resources required for testing was achieved by providing a reliable testing environment accessible to all entities working on the project and fully automated testing pipelines. In addition to that, the VT framework allows to start with the module development earlier in the project timeline which has a significant impact on the project budget.

6.3 Evaluation summary

The evaluation of the VT framework based on the feedback of two project teams suggests that the project goals and requirements were successfully fulfilled. The VT framework is easily applicable for new projects within the given time frame and considering the feedback from the developers it is easy to use on their local machines.

In terms of compatibility, the VT library provides a comprehensive interface that allows to control of the tested module to any testing client that implements the required interface functions. Thanks to this approach, the framework can be used across all Company's locations where the I/O modules are being developed.

One of the crucial requirements was to improve the quality of testing in terms of module control and frequency to lower the necessary resources for testing by avoiding the restarting of the testing process on the physical hardware. That was successfully achieved by the integration to the CI/CD pipeline, precise module control, and Visual Studio debugger.

The CI/CD pipeline verifies the functionality of any newly implemented code once it is added to the repository. The virtual testing was added to already existing test steps such as Unit testing or Static Code Analysis. Thanks to the automated testing in the testing pipeline the bugs are discovered before other features are based on their behavior and the bug fixing becomes a complex task.

The precise module control allows to verification of various edge cases that are difficult to achieve on the physical hardware. The testing client can execute the technology code for any given time beginning from 1 nanosecond and request the data from the module to verify if it is correct. In case the data stored in the data objects is not enough for the user, there is a possibility to debug the code line by line by using the Visual Studio debugger. Thanks to these features, the VT framework provides a type of gray box testing that is not possible to achieve on the physical hardware and was currently missing in the Company's testing strategy.

The improved quality of testing and the possibility to start with the development earlier in the project timeline has a significant impact on the project's budget and lowers the necessary resources for testing, which was the main goal of the project. Besides the positive impact on the project's timeline, the project managers in the Company appreciate the better predictability of the development process. That allows them to plan the resources more efficiently and avoid the situation when the developers are unable to work on the project they are assigned to.

To sum it up, the VT framework fulfilled all the requirements and goals set in section 3.2.4. The project is considered successful from all perspectives and is likely to become part of the testing strategy in the Company. The VT framework is a valuable tool for developers and testers that brings various benefits described above. Besides that, the usage of virtual testing offers better predictability to the module development which is highly appreciated by the project managers as it allows them to plan their resources more efficiently. In addition, the initial investment is relatively small and therefore even small projects are likely to benefit from the adaptation of the virtual testing.

6.4 Suggestions for future improvements

Despite the success of the VT framework project, there is room for improvement. This section aims to describe what steps would improve the usability of the framework in the Company and improve its impact on the module development. The suggestions are based on the evaluation of the VT framework performed in the previous sections.

The first suggestion is to switch from the Visual Studio solutions to the CMake build system. The Visual Studio solutions are not compatible with the Linux operating system. That can be problematic in the future for various reasons. Firstly, the developers often use docker containers to prepare the environment for specific projects and test cases. Since the docker containers are usually based on the Linux operating system, the VT framework cannot be currently used for this purpose. Secondly, many developers in the Company prefer to use the Linux operating system for development, which is allowed if does not interfere with the project requirements. Therefore, it would be

inconvenient for many developers if the only reason they need to switch to the Windows operating system is the VT framework.

Another suggestion is to increase the modularization of the VT library. If the VT library is divided into smaller independent components, it would be easier to maintain and extend the framework in the future. It would also allow to more easily adapt the VT framework for new projects as the developers could choose only the components they need for their specific project. Besides that, the modularization would allow sharing of the specific components across multiple projects that use the VT framework and therefore further reduce the initial investment required for the VT framework adaptation.

Lastly, the server application currently relies on Windows inter-process communication. The Windows pipelines are very limiting in terms of the number of processes that can be created and the number of messages that can be sent. Besides that, the Windows pipelines are not compatible with the Linux operating system, which could be problematic in the future. Therefore, it is suggested to switch to platform-independent socket communication that would allow communication between the server application and the testing client on any operating system.

In summary, after the evaluation of the VT framework, the following improvements should be considered to improve the usability of the VT framework in the Company:

- Switch from Visual Studio solutions to the CMake build system
- Increase the modularization of the VT library
- Switch to platform-independent socket communication

These improvements will enhance the accessibility of the VT framework for developers and testers across all Company locations. They will also reduce the initial investment required for the adaptation of the VT framework. Moreover, by modularizing the framework, specific components can be shared across multiple projects, reducing the risk of bugs in the implementation of the VT framework and reusing already tested components. It is expected that the reusability of the components could significantly improve the user experience and convince the project managers at other locations to adapt the VT framework for their projects.

Conclusion

The main goal of this thesis was to propose and develop a Virtual Testing Framework that would improve the efficiency of the current testing process in the Company and therefore reduce the time and resources required for testing. To achieve this goal, numerous steps needed to be taken.

Firstly, get familiar with the current process of testing throughout the development process in the Company and understand the testing requirements and the overall workflow of the testing process in the Company. This process is described in section 2.4 focusing on the testing process in section 2.4.3.

Secondly, analyze the current testing and implementation process to identify areas for improvement in resource management. The analysis is performed and described in chapter 3, especially in section 3.1.

Thirdly, derive the requirements for the Virtual Testing Framework based on the identified inefficiencies. The requirements are described in section 3.2. Once the requirements were identified, the Virtual Testing Framework was proposed, designed, and described in section 3.3.

Fourthly, implement the Virtual Testing Framework according to the proposed architecture, design and requirements. The implementation is described in chapter 4. The functionality of the Virtual Testing Framework is then demonstrated in chapter 5.

Finally, evaluate the Virtual Testing Framework and its impact on the project delivery process. The analysis of the impact is performed 3.4. The follow-up evaluation is then described in chapter 6, which is based on the feedback from the developers, testers and project managers.

Despite the relatively complex requirements that were set for this project due to the many different entities that are involved in the testing process, the Virtual Testing Framework was successfully implemented. The provided feedback suggests that the framework has a positive financial impact on the projects and increases the predictability of the project timelines which is highly appreciated by the project managers.

In conclusion, all objectives of this thesis have been achieved. The proposed solution has met all the requirements and has been successfully integrated into the development process of the Company. Despite the success of the project, there is still room for improvement and further development such as ensuring platform independence by switching to a Cmake build system and TPC/IP socket communication.

Bibliography

1. MACHINEMETRICS. *I/O modules: Enabling device connectivity and control* [online]. 2023. Available also from: https://www.machinemetrics.com/connectivity/hardware/io-modules. [Cited 03-04-2024].
2. *Siemens industry mall - I/O modules* [online]. [N.d.]. Available also from: https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10046659. [Cited 03-04-2024].
3. WESTLAND, Jason. *The Project Management Life Cycle A Complete Step-By-Step Methodology for Initiating, Planning, Executing Closing a Project Success*. Kogan Page, 2007. ISBN 9780749448080.
4. ASANA, Team. *What is IT project management? [2023] • asana* [online]. Asana, 2022. Available also from: https://asana.com/resources/it-project-management.
5. COLE, Ben. *What is IT project management?: Definition from TechTarget* [online]. TechTarget, 2015. Available also from: https://www.techtarget.com/searchcio/definition/IT-project-management. [Cited 2024-06-01].
6. BRIDGES, Jennifer. *What is a feasibility study? how to conduct one for your project* [online]. ProjectManager, 2023. Available also from: https://www.projectmanager.com/training/how-to-conduct-a-feasibility-study. [Cited 2024-06-01].
7. AKANKSHA VERMA Amita Khatana, Sarika Chaudhary. *A Comparative Study of Black Box Testing and White Box Testing*. IJCSE, Indore, INDIA, 2017. Available from DOI: https://doi.org/10.26438/ijcse/v5i12.301304.
8. PHANIVEDALA. *Black Box Testing : Exploring types and benefits* [online]. 2023. Available also from: https://www.extnoc.com/learn/security/black-box-testing. [Cited 2024-31-01].
9. SAMRA, Hardeep. Study on Non Functional Software Testing. *INTERNATIONAL JOURNAL OF COMPUTERS AND TECHNOLOGY*. 2005, vol. 4, pp. 151–155. Available also from: https://www.researchgate.net/publication/324985802_Study_on_Non_Functional_Software_Testing. [Cited 2024-31-01].

10. TOMAR, Vinita; BANSAL, Mamta; SINGH, Pooja. *2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST)*. Regression Testing Approaches, Tools, and Applications in Various Environments. 2022. Available from DOI: 10.1109/AIST55798.2022.10064753. [Cited 2024-31-01].

11. BRAR, Hanmeet Kaur; KAUR, Puneet Jai. *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. Differentiating Integration Testing and unit testing [online]. 2015. Available also from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7100358.

12. AWATI, Rahul. *What is integration testing (IT)?* [Online]. TechTarget, 2022. Available also from: %5Curl%7Bhttps://www.techtarget.com/searchsoftwarequality/definition/integration-testing%7D. [Cited 2024-03-02].

13. KAUR, Paramjeet. A Research Paper on White Box Testing. *International Journal* [online]. 2018. Available also from: https://wwjmrd.com/upload/a-research-paper-on-white-box-testing_1519477825.pdf. [Cited 2024-03-02].

14. *What is unit testing?* [Online]. AWS, 2023. Available also from: https://aws.amazon.com/what-is/unit-testing/. [Cited 2024-05-02].

15. MIECZNIK, Rafal. *The importance and benefits of unit testing* [online]. CodiLime, 2023. Available also from: https://codilime.com/blog/unit-testing/. [Cited 2024-05-02].

16. RANA, Kuldeep. *Unit testing: Definition, advantages and limitations* [online]. 2023. Available also from: https://artoftesting.com/unit-testing. [Cited 2024-05-02].

17. GILLIS, Alexander S. *What is Static Analysis (Static Code Analysis)?* [Online]. TechTarget, 2020. Available also from: https://www.techtarget.com/whatis/definition/static-analysis-static-code-analysis. [Cited 24-02-2024].

18. KANSARA, Darshil. *What is static code analysis and how it works? understanding sast in DevOps* [online]. Radixweb, 2023. Available also from: https://radixweb.com/blog/what-is-static-code-analysis. [Cited 24-02-2024].

19. KAUR, Paramjeet. *A Research Paper on White Box Testing* [online]. 2018. Available also from: https://wwjmrd.com/upload/a-research-paper-on-white-box-testing_1519477825.pdf. [Cited 25-02-2024].

20. MOHD EHMER, Farmeena Khan. *A Comparative Study of White Box, Black Box and Grey Box Testing Techniques* [online]. 2012. Available from DOI: 10.14569/IJACSA.2012.030603. [Cited 2024-17-02].

21. GFG. *Gray box testing - software testing* [online]. GeeksforGeeks, 2023. Available also from: https://www.geeksforgeeks.org/gray-box-testing-software-testing/. [Cited 2024-17-02].

22. SAXENA, Ruhi; SINGH, Monika. *Gray Box Testing: Proactive Methodology for the Future Design of Test Cases to Reduce Overall System Cost* [online]. Krishi Sanskriti Publications, 2014. ISSN 2350-0077. Available also from: http://www.krishisanskriti.org/jbaer.html. [Cited 2024-20-02].

23. BRUSH, Kate; KIRSCH, Brian. *What is virtualization? definition from searchservervirtualization* [online]. TechTarget, 2021. Available also from: https://www.techtarget.com/searchitoperations/definition/virtualization. [Cited 04-04-2024].

24. ANNAMALAI, Banushri; KARTHIKA, R.A. Implementation levels of virtualization and security issues in cloud computing. *International Journal of Engineering and Technology(UAE)* [online]. 2018, vol. 7, pp. 678–682. Available also from: https://www.researchgate.net/publication/326683646_Implementation_levels_of_virtualization_and_security_issues_in_cloud_computing. [Cited 04-04-2024].

25. VMWARE. *What is application virtualization? | vmware glossary* [online]. 2023. Available also from: https://www.vmware.com/topics/glossary/content/application-virtualization.html. [Cited 04-04-2024].

26. JAVAPOINT. *Operating system based Virtualization* [online]. 2022. Available also from: https://www.javatpoint.com/operating-system-based-virtualization. [Cited 04-04-2024].

27. LENOVO. *What is a Hardware Abstraction Layer and How Does it Work? | Lenovo US* [online]. 2023. Available also from: https://www.lenovo.com/us/en/glossary/hardware-abstraction-layer/?orgRef=https%253A%252F%252Fwww.google.com%252F. [Cited 04-04-2024].

28. ARM LTD. *What is Instruction Set Architecture (ISA)?* [Online]. [N.d.]. Available also from: https://www.arm.com/glossary/isa. [Cited 04-04-2024].

Appendix A

Acronyms

API Application Programming Interface

DI Digital Input

DLL Dynamic-link library

FTE Full-time employee

HAL Hardware Abstraction Layer

HW Hardware

IPC Inter-Process Communication

ISA Instruction Set Architecture

OS Operating System

QI Quality Information

ROI Return on Investment

PLM Project Management Lifecycle

SC Short Circuit

SW Software

TCP/IP Transmission Control Protocol/Internet Protocol

ToR Terms of Reference

UT Unit Testing

WB Wire Break

WBS Work Breakdown Structure

Appendix B

User Manual

B.1 Prerequisites

To run the Virtual Testing system for the sample module project included in the attachment of this thesis you need to have the following prerequisites installed on your computer: Visual Studio 2019 and Python 3.10 or newer. The system was tested on Windows 10, but there are no apparent reasons why it should not work on different Windows versions.

B.2 Project preparation

The structure of the project is shown in the attachment contents in Chapter C. To run the Virtual Testing framework adjusted for the sample module project, you need to follow these steps:

1. Open the emul_dll project in Visual Studio 2019 or newer and build it (Debug, x86 platform)

2. Open the emul_app project in Visual Studio 2019 or newer and build it (Debug, x86 platform)

3. Open the module_client project in Visual Studio 2019 - No need to build it

B.3 Running the Virtual Testing system

Everything should be prepared for running the Virtual Testing at this point. To run the test cases prepared for the sample module project, follow these two simple steps:

1. Run the emul_dll project (The project is set to run the emul_app server application with the required command line arguments)

2. Run any of the test cases included in the module_client project

The comment line arguments for the emul_app server application can be adjusted in the project properties by right-clicking on the project in the Solution Explorer and selecting Properties, then navigating to the Debug tab. The default arguments are set as follows:

```
-L ..\..\emul_dll\Debug\emul_dll.dll -I --log-level INFO
```

The most common change would be to adjust the logging level.

In case of the need to run multiple test cases, there is a possibility to adjust the test_all.py script, which is currently set to run all the test cases included in the project.

Appendix C

Attachments Contents

- `readme.md` — Contents description
- `emulator` — Emulator project
  - `emul_app` — Server application for emulator
  - `emul_client_py` — Shared files for VT clients
  - `emul_common` — Files shared between VT library components
    - `templates` — Shared templates
    - `util` — Shared tools
  - `emul_dll` — Emulator DLL library
    - `dll_llc_layer` — LLC layer
    - `emul_core` — Emulator core - Event calendar
    - `emul_log` — Shared logging
    - `emul_SPI` — SPI emulation
    - `module_data_obj` — Data objects for the sample module
    - `module_interfaces` — Module interface
    - `module_client` — Directory for testing clients
      - `sample_module_client` — Client adjusted for the sample module
- `module_core` — Module layer controlling hardware
- `src` — Source code of the module - Technology code
- `Thesis.pdf` — PDF version of the thesis