Tampere University

Jussi Joenperä

AUTOMATED SOFTWARE TESTING OF VENTILATION EQUIPMENT

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Supervisor: Hannu Koivisto, Mikko Salmenperä
February 2022

ABSTRACT

Jussi Joenperä: Automated software testing of ventilation equipment
Master of Science Thesis
Tampere University
Master's Programme in Automation Engineering
February 2022

In product development, testing is a vital process to ensure the product's quality. Testing takes a lot of time and resources to be effective. Usually, these are all repetitive tasks and are prone to mistakes. Automation can be a solution for these problems. Automating tests can be resource heavy activity, so planning it is necessary to ensure its success. Automation is not a silver bullet because not everything could or should be automated.

In this thesis, manually executed equipment tests are automated. Designing these automated tests takes precedence over the implementation. The focus in on three devices that belong to a company called FläktGroup. One of the company's business areas is the development and testing of ventilation and fire safety equipment. The company is interested in automating these devices' tests.

First, a literature review of software and hardware testing and test automation was conducted to understand testing process and automating tests. A review of the equipment and their current tests was conducted to get a better understanding of them. After this, the tests were analysed to detect their possibilities and limitations for automation. These two steps included inspecting the equipment, reviewing their technical data, and interviewing testers. From collected information new automated tests were designed. This included studying previously used and completely new resources. Only one devices' automated tests were implemented. Some steps in test process could not be automated due to the technical and time limits.

The results consist of a few design options for test automation. Designs for two devices gave promising results. Each design option was different in terms of used software. The wide range of software options allows comparison for the designs, and thus the best solution can be picked for each device. Some designs use same resources as in manual tests, but increasing automation requires new software. Improvements were gathered for testing process and new tests. These include software upgrades and documentation. Overall, the results gave a good start for test automation.

Keywords: Modbus, FläktGroup, software testing, test automation, ventilation equipment

TIIVISTELMÄ

Jussi Joenperä: Ilmanvaihtolailleiden automatisoitu ohjelmistotestaus
Diplomityö
Tampereen yliopisto
Degree Programme
Helmikuu 2022

Tuotekehityksessä testaus on tärkeä prosessi tuotteen laadun varmistamiseksi. Testaus vaatii paljon aikaa ja resursseja ollakseen tehokas. Yleensä nämä kaikki ovat toistuvia tehtäviä ja ovat alttiita virheille. Automaatio voi olla ratkaisu näihin ongelmiin. Testien automatisointi voi olla resurssiraskasta toimintaa, joten suunnittelu on välttämätöntä sen onnistumisen varmistamiseksi. Automaatio ei ole hopealuoti, koska kaikkea ei voi tai pitäisi automatisoida.

Tässä opinnäytetyössä manuaalisesti suoritetut laitetestit automatisoidaan. Näiden automatisoitujen testien suunnittelu menee tärkeysjärjestyksessä testien toteutuksen edelle. Työssä keskitytään kolmeen laitteeseen, jotka kuuluvat FläktGroup-nimiselle yritykselle. Yrityksen yksi toimialoista on ilmanvaihto- ja paloturvallisuuslaitteiden kehitys ja testaus. Näiden laitetestien automatisointi kiinnostaa yhtiötä.

Ensin suoritettiin kirjallisuuskatsaus ohjelmistojen ja laitteistojen testauksesta sekä testiautomaatiosta testausprosessin ja testien automatisoinnin ymmärtämiseksi. Laitteita ja niiden nykyisiä testejä tarkasteltiin, jotta niistä saataisiin parempi käsitys. Tämän jälkeen testit analysoitiin, jotta saataisiin selville niiden mahdollisuudet ja rajoitukset automatisoinnissa. Nämä kaksi vaihetta sisälsivät laitteiden tutkiminen, teknisten tietojen tarkastelu ja testaajien haastattelu. Kerätyistä tiedoista suunniteltiin uusia automatisoituja testejä. Tämä sisälsi aiemmin käytettyjen ja täysin uusien resurssien tutkiminen. Vain yhden laitteen automatisoidut testit toteutettiin. Joitakin testiprosessin vaiheita ei voitu automatisoida teknisten ja aikarajoitusten vuoksi.

Tulokset koostuvat muutamasta suunnitteluvaihtoehdosta testiautomaatiolle. Kahden laitteen suunnitelmat antoivat lupaavia tuloksia. Jokainen suunnitteluvaihtoehto oli erilainen käytettyjen ohjelmistojen suhteen. Laaja valikoima ohjelmistoja mahdollistaa suunnitelmien vertailemisen, jolloin jokaiselle laitteelle voidaan valita paras ratkaisu. Jotkut suunnitelmat käyttävät samoja resursseja kuin manuaalisessa testauksessa, mutta kasvava automaatio vaatii uusia ohjelmistoja. Parannusehdotuksia kerättiin testausprosessia ja uusia testejä varten. Näihin kuuluvat ohjelmistopäivitykset ja dokumentaatio. Kaiken kaikkiaan lopputulokset antoivat hyvän alun testiautomaatiolle.

Avainsanat: Modbus, FläktGroup, ohjelmistotestaus, testiautomaatio, ilmanvaihtolaite

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
Modbus	master-slave communication protocol
RTU	Remote Terminal Unit
UML	Unified Modeling Language
VAV	Variable air volume

1. INTRODUCTION

Testing is one of the most important processes when it comes to quality assurance of equipment. Testing software and hardware manually is time consuming process and repetitive. This can lead to errors and waste of time because verifying results manually takes time. These problems can be fixed by automating some tests.

FläktGroup is a company that specializes in ventilation and fire safety equipment. The company develops, tests, and produces equipment and systems for these purposes. Equipment has important, some of which are lifesaving, tasks so testing them thoroughly is necessary. The company attaches great importance to the safety of the equipment. Any new or updated product needs to be tested before being released to the market. Most of the equipment tests are done manually. The company wants to automate some of the testing. From the company's equipment range three devices are chosen for this thesis.

1.1 Goal

A goal is created to make it easier to approach the thesis's topic. The goal of this thesis is to improve testing of equipment by automating some manually done tests. Previously testing of the equipment have been all done manually. This has bound employees full time.

The goal is divided into four subgoals. This way the goal is easier to understand and with smaller objectives thesis's basic structure is created. They help to divide the work into segments and give them clear targets. Success of each subgoal depends on previous ones' end results. The subgoals are

- Familiarize oneself with the equipment and their tests
- Analyse old tests
- Design automated tests
- Implement automated tests

First subgoal is about gathering information about the equipment and their tests. This includes technical reviews, inspecting physical devices, and interviewing testers. Second

subgoal is about analysing the tests. Their results should give what exactly needs to be automated.

Third subgoal is about the design of test automation. This includes studying and considering possible software and hardware for new tests. The fourth subgoal, implementation, is about creating designed tests. The subgoal is not necessarily achievable in some cases due to the time and technical limits. Three devices give enough work for one thesis. Thesis' structure and order follow these subgoals.

1.2 Thesis structure

Thesis starts with this Introduction. This is followed by chapter 2 where the basics of software and hardware testing are explained. The chapter includes a literature review of testing approaches, levels and testing types. Only those testing types that are used in this thesis are mentioned. Chapter 3 focuses on describing equipment that is subject to testing. The equipment is divided into three levels, each containing one device. This partition is done because each level is developed by different set of companies. These equipment levels and their devices are

- Level 1: Ultra Ulsa, a VAV damper
- Level 2: Fico Pro, a fire damper control and monitoring system
- Level 3: Ultrasound air flow measurement device

Chapter 4 explains the analysis of tests, design of automated tests, and their implementation. First part of the chapter shows analysis of manually done tests and how they should be prioritized. Second part of the chapter explains how tests could be automated for each type of equipment type. The last part shows the implementation of automated tests.

Chapter 5 shows the results of created tests. The compliance with requirements, possible future improvements for testing equipment and personal thoughts are addressed in the chapter. Chapter 6 summarizes the main findings and conclusions.

2. TESTING

Testing ensures the system works like it's required to work. All functionalities should follow the planned designs. Today software and hardware testing is a major part to ensure the quality of the system. When size and complexity of system grows, importance of testing becomes more crucial.

This chapter starts with a description of testing levels. This means there are different test types at various phases of software project. This part of chapter introduces a V-model used in software development. Second part of this chapter is about the basics of testing process. This process is used from the start of the software project and used until the project is finished. Third part focuses on how to approach testing and test planning. Fourth part focuses on automated testing. In there the reasons and strategies for automation are explained. The last part of this chapter is about hardware in the loop, a testing technique where some part of system is replaced by a simulation.

2.1 Levels

Sequential development model is a software development model where different parts of the system are done in sequence. First the objectives are defined, then design of components in defined sequence. After requirements are checked, the system should be ready. An old model of this type was waterfall model where next development phase can only begin after the previous one has been completed. Problem with this model is the testing is done at the end of project. Another problem with this model is a defect that is created in one and detected in later phase. Defect should be detected in their own phases. Upgraded version to this mode is V-model. Testers should learn this model and how to use it even if the software project they are in uses other type of development model. The basics of V-model, even if used sparsely, can be applied. [1] [2]

V-model is a development model where testing is taken as a major part of software development. The model explained first because after that the testing levels are cleaner. Figure 2.1 shows the structure of v-model. V-model is also known as Verification, validation, and acceptance model. Sequential model to group levels together. V shape of the model help to visualize verification and validation phases and how they are connected. Left side of the model is defining system. Each phase is completed in descending order.

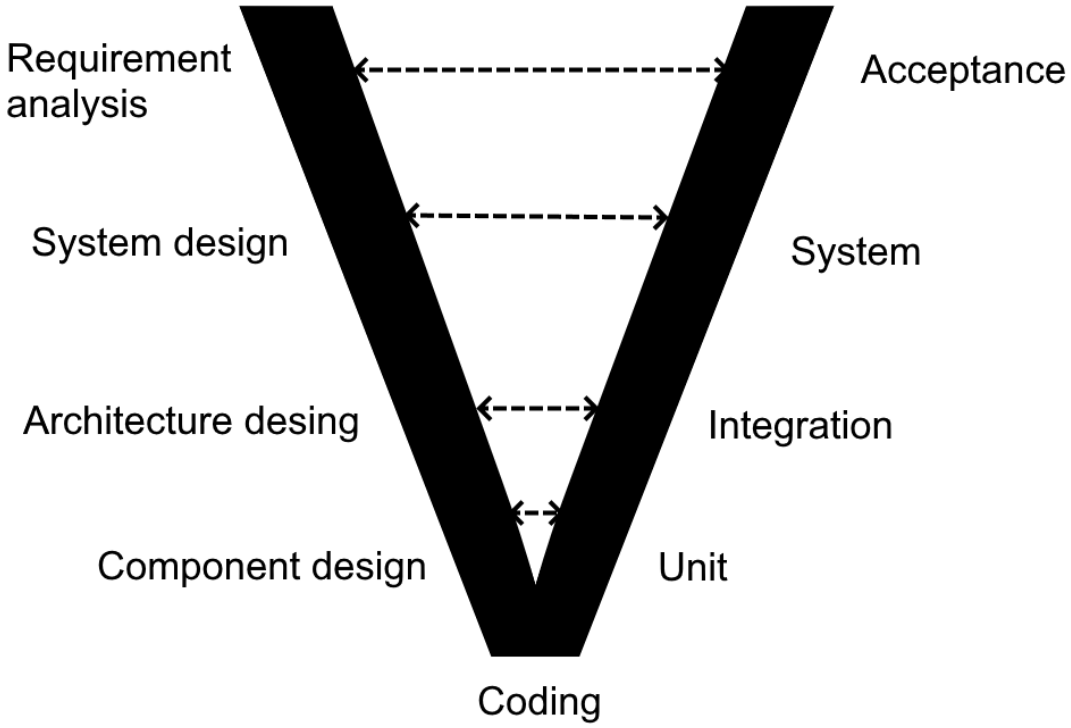

Figure 2.1. *Development and testing phases in V-model*

Right side is for verification phases. These phases are also known as testing levels. Each phase in this side of the models are completed in ascending order. Tests are developed to comply with design choices. A testing phase exists for each design phase. This means every level of design and requirements are tested. [3]

The model is divided to several phases. These phases are verification, coding, and validation phases. Coding phase includes creation of components, integrating them together to create whole software system parts, and finally join these parts together to create whole system. Verification phases ensure that each part of the system meets its specifications. Verification answers the question of whether the system was built right. There are four verification phases: requirements definition, system design, architecture design and component design. The flow of verification goes in this direction. This way the system is design from top to bottom and built from bottom to top. [1] [4]

Requirements' definition is done by customer and end-user. Main objective of this phase is what things are wanted and excepted from the system. This gives goals to the software project. Acceptance tests are developed by outcomes of this phase. System design, also known as functional design, is a phase where requirements of specific features and system functions are defined. System tests are developed by outcomes of this phase. [1] [4]

Architecture design contains defining required interfaces to the world outside the system. The system is divided to subsystems, and they are divided to components. This design is high-level designing in software development. Integration tests are developed by outcomes of this phase. Component design, also called module design, is a phase where behaviour, structure and tasks of components are defined. Unit tests are developed by outcomes of this phase. [1] [4] [3]

Validation phases ensure that the system works as well as planned. This includes testing every part of the piece and entirety of the system, from the smallest component to system verification. Validation answers to question of whether the right system was built. Validation phases, also known as testing levels, are the cornerstone of software testing. Typically software testing is done at four different levels: unit, integration, system and acceptance testings. Validation starts with unit and ends with acceptance testing. Testing can only proceed to next level after the previous ones have been completed. Testing levels are not used in every system. Web development often uses different kind of level system because time between implementing components and integration can be such a short time. [1] [4] [5]

Unit testing stands for testing the smallest piece of system called unit. A unit can be a function, component, database request or module. Objective of this type of testing is to detect failures and verify the right functionality. This includes considering behaviour of input values and how they meet requirements. Unit testing is mostly white box testing because the insides of system are known and tested. testing could be black box testing if tester only has access to unit interfaces. Unit testing detects the most bugs found in testing. It is important to invest time for testing because of this. All defects multiply when going up in system complexity. [1] [5]

Integration means units are combined together to form bigger part of the system. These parts are called subsystems and integration stops when the whole system is assembled. At least two components must communicate with each other to be called subsystem. There are multiple ways to integrate components. The integration approach depends on fragmentation of components. [4]

Integration testing is a phase where functionality of assembled subsystems are tested. System components could work perfectly fine individually but they could cause defects when they work together. The purpose of testing is to find failures in the interfaces and communications between components. Integrity is one type of testing the maturity of system. Integration testing is black box testing if tester can only read subsystem specifications and access interfaces. [1] [4] [5]

System testing is done after integration and therefore after the system is completed. After assembling subsystems together to create the system, system testing is done. The purpose of this testing level is to detect failures in the system and verify the system fol-

lows requirements and specifications. These failures should be detected at the highest level of integration. User tests is crucial part of this level when testing user interfaces. Most of tests of this level are black box testing because they are based on specifications and testers cannot manipulate system's inner workings. In unit and integration testing the specifications are checked for developers' standpoint and in system testing the focus is on customer's and users' viewpoint. Basically low level testing focuses on details and system testing focuses on the whole system. [5] [2]

Acceptance testing is the last phase of verification. The objective of these tests is to get customer's and user's acceptance. Before starting this level the system testing level must be completed. This means that the system is considered completed product and ready to be delivered. Founding failures is not the main objective but to gain confidence in the system. tests are executed by customer, not by previous testers. They perform system wide tests to test their requirements. In the end customer or end-user testing accept the system as a ready product. [1] [5] [2]

Like every development model, the v-model has its strengths and weaknesses. First a few good things about the model. Each phase is well defined. Easy to use and implement. When a defect is detected, the whole tests process can be improved so that the same type of defects can be detected earlier. Then there are the weaknesses, the reasons why some use alternative agile models. The v-model becomes pointless in complex and long-term software projects. From small to big: no working software, nothing to show to clients and users. Not suitable for agile projects, requirements keep changing. Validation phases are not always good enough. In the model test design should happen at the same time as the development. In reality testing often happens after the development. [1] [5] [2]

2.2 Process

Testing is major part of software development and should not be taken lightly. Executing tests is just a small part of testing process and multiple other phases happen before testing itself. The main goal of testing is to improve the quality of the system. It is important to focus on quality rather than quantity of tests. Testing process can be divided to four different phases. Figure 2.2 shows these phases and their execution order. The implementation phase is divided to four smaller phases. [6]

Test planning is the first part of testing process. Planning answers the most important questions of what things are being tested and how tests' success is measured. Using software requirements and prioritizing them to establish is a good starting point for tests. The timing of tests is desirable to have been planned before the project has progressed much. This means deciding what kinds of tests are executed at each stage of the project. Choosing test creators and testers influences employee resources because software test-

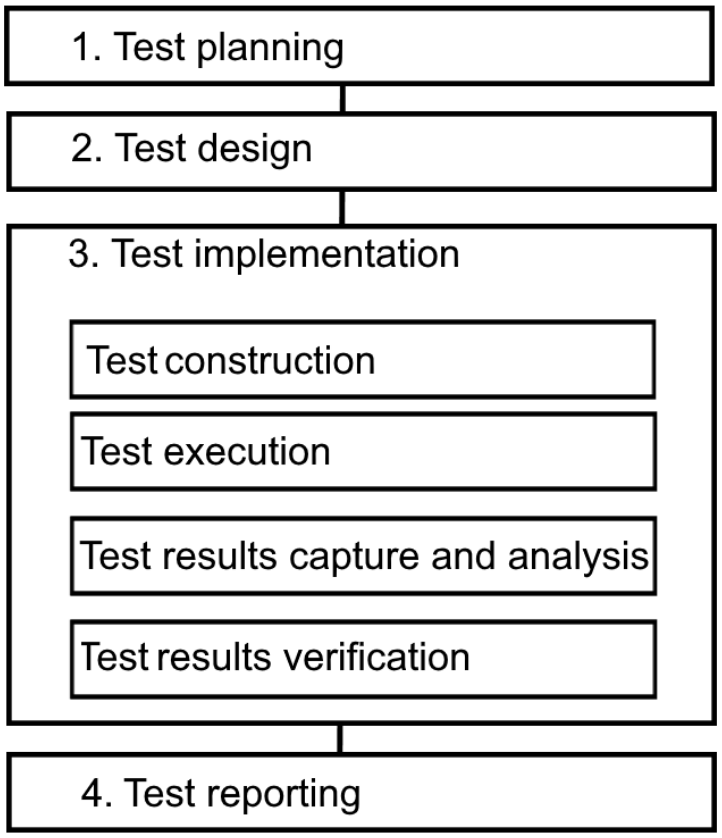

Figure 2.2. *Phases of testing process*

ing is time consuming. Testing approaches and levels help to plan tests. More about these can be found later in this chapter. The end product of this phase is test plan that includes test case requirements. The test plan is used to construct test cases. It is updated if requirements change during the production. The planning should be done throughout so that reversing back to this phase because of errors is kept minimum. The phase is visited often during the development because the project's requirements tend to change. [6]

Test design comes after test planning and uses test requirements from that phase. Last phase was about finding things to test, and in this phase the main point is to find how they are tested. This includes identifying test conditions that follow test plan. The conditions should take into account the behaviour of test cases. Examining the test case behaviour creates multiple tests. Design phase produces tests and test data. Because they depend on requirements, they should be reviewed when updating the test plan. Test cases are created from test case requirements. Test data refers to values and data types that are used in testing. The data is used to check if test case follow the excepted behaviours. Designing might be the most important part of testing because it affects every other phase. It's also the most time consuming phase. [2] [6]

Test implementation is where tests are created, executed and test results are gathered. The implementation follows the end results of design phase. Implementation phase is

divided to four different sub-phases. These sub-phases are test construction, test execution, test results capture and analysis and test result verification. The partition is done because each sub-phase has its own characteristics. Test construction is the first sub-phase of implementation. In this sub-phase designed tests are created. Creating test scripts and test data are part of this sub-phase. The scripts are part of test automation. Instructions are created for manual tester. This document could be an excel file that the tester fills during tests. The sub-phase uses planned tools from design phase. [7] [2] [6]

Test execution comes when tests are constructed. This sub-phase can be done by manually or it can be automated. Using both methods is desirable because they find different types of errors. Automation takes effort but saves a lot of time. It should be used whenever possible. Test results capture and analysis is about logging test results and draw conclusions from them. Tests are logged as pass, fail, or skipped. Reason for failures and skips are also logged. Automated tests log their own test results. Test logs should include information of what was tested or why it was not tested, when, by whom, and with what results. The format and structure of these logs should be modelled in design phase. Test results verification is the last part of test implementation. In this sub-phase the gathered results are compared to excepted output data. Use of test logs is critical to verify the results. Verification can be done manually or with test automation. Manual checking is slower of these two but necessary if verification needs subjectivity inspection. This is useful when verification is hard to automate due to the requirements for verification. The sub-phase includes comparing old test results to new ones. This is done to see the progress of testing and if new tests are following their requirements. After implementation phases comes reporting of tests. [2] [6]

Test reporting is the last phase of testing process. In this phase the test results and their analysis are documented to give a picture of the quality of the system. Different report types exist for various purposes. Some reports are created multiple times during the project, and some are created only at the end of it. There are two the most important types of reports, each having their own audience and goals. First one is test summary report is that is meant for technical managers and customers. This document gives an estimation of product quality and if the product can be approved as ready. Second important report is detailed report that is created for the project's development team. This document includes detailed description of performed tests. It also has description of progress of testing development. [2] [6]

There are four different terms when it comes to identifying incorrect outputs and defects. Mistake refers to incorrect result created by human action. It is also known as a human error. Fault is an incorrect step, process, or data definition in a program. Terms error, bug and defect are commonly used. For example, an incorrect instruction and statement are faults. Error is the most common when. Error happens when output value differs from planned, specified, or correct value. In short, the real output is compared to planned one.

The last term, failure refers to system's or its component's inability to fulfil its required functions within its performance requirements. [3]

2.3 Approaches

Testing is divided to two different categories, white box and black box testings. These two decide the basic approach to testing. The approaches show the point of view from system to testers. This gives direction on designing test cases. This section of chapter gives information about these testing approaches and common techniques on how to utilize them. [7]

In white box testing the system is seen as seen through box because testers can see inside of the box. Internal workings, structures and logic of programs are known. Purpose of testing is to ensure proper functioning of system and all parts are used. All paths created by logical conditions are to be tested. This path testing includes coverage, correct directions, and variable handling. [7]

There are three common white box testing techniques. These are code coverage, control structure testing and loop testing. Each technique brush same code parts but test different thing. In code coverage testing segments of code control structure are executed at least once. All possible paths in the code are run through. Paths are created when a if else condition is met in code. Control structure testing is divided to two parts. First part is condition testing. Here logical conditions are tested so that the conditions channel different values of variables to correct paths. For example, a function may trigger one path to integer variable that is equal or less than zero and other path for variables that are greater than zero. Second part of control structure is data flow testing where program's variables are analysed. This includes variables' definition and changes during the program execution. Loop testing focuses on testing loops in code. Its objective is to examine different kind of loops, their performance, behaviour, structure, and loop amount. This way it is easier to improve and fix loops. [7]

In black box testing the system is seen as black box because the system's inner workings are unknown. Test are executed using only interfaces that are available to user. Only functionality, inputs and expected outputs are known. Real outputs are compared to excepted outputs. In short, the tests are as user. Black box testing requires knowledge of system requirements. This is to ensure every part of the system works as intended. Negative testing has to be employed to test system's error handling. This means the tester gives invalid inputs to see if system can handle these inputs. [1] [4] [7]

There are three common black box testing techniques. These are equivalence partition, boundary value analysis and decision tables. Because of near infinite amount of different input options there are near infinite number of test cases the system can be tested

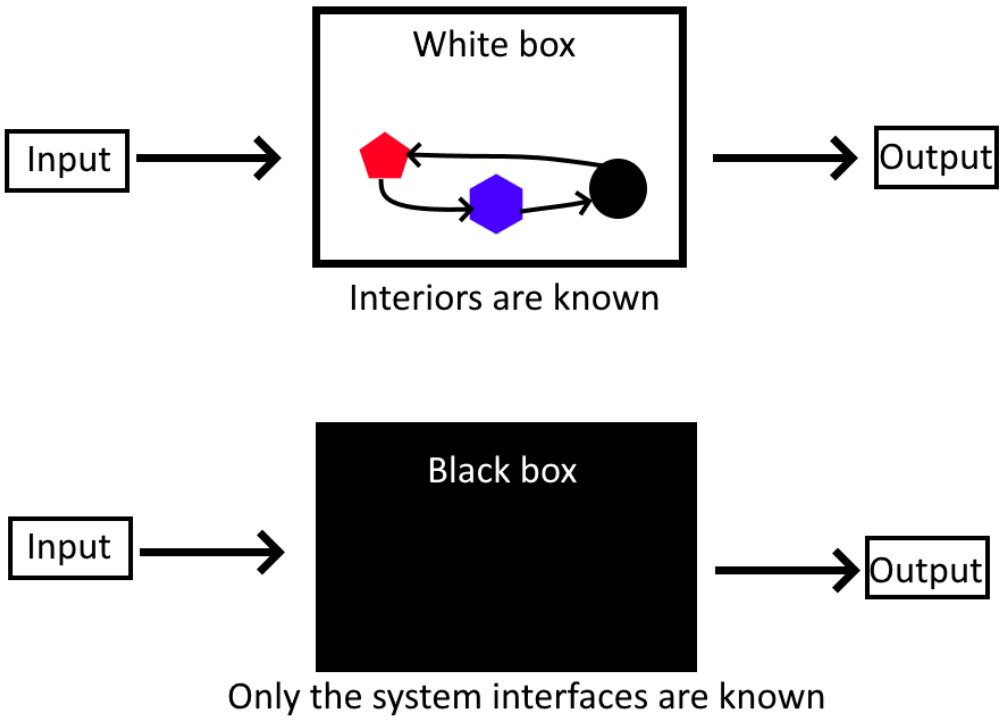

Figure 2.3. *Differences of black and white box testing*

against. In equivalence partition inputs are separated in equivalence classes. Values in each class presents all members of the class. This separation means that the system handles every value in class in a same way. There a several ways how values are divided to equivalence classes. One way is to divide inputs to valid and invalid classes. Other way is to divide values on how logical conditions react on them. For example, a program calculates area of circle. Circle's radius is input. Input values would be divided to two classes: valid numbers that are greater than zero and invalid numbers that are equal or less than zero. Former class could be divided to smaller ones if the calculations use some thresholds for areas. Another example would be a class that handles date times. Parameters of this class are day of the month, month, and year. Days would have a range of valid values between one and thirty-one. Months have different number of days so this would create new equivalence classes. [1] [4] [7]

Boundary value analysis is a next step after getting classes from equivalence partition. The analysis focuses on boundaries of classes. It is analysed how system behaves near these boundaries. Test values are chosen from above and below these boundaries to test if limits of these values are coded correctly. This means minimum, less than minimum, maximum, and greater than maximum values are testes. This works best when there are none or minimal amount of correlation between parameters. The analysis is a good technique to test that all unequal signs are correct in comparisons. Using previous

	A	B	C	D	E
1		**Language**	**Browser**	**Font size**	**Screen size**
2	**1**	English	Firefox	Small	PC
3	**2**	English	Chrome	Normal	Laptop
4	**3**	English	IE	Big	Phone
5	**4**	Spanish	Chrome	Big	PC
6	**5**	Spanish	IE	Small	Laptop
7	**6**	Spanish	Firefox	Normal	Phone
8	**7**	English	IE	Normal	PC
9	**8**	English	Firefox	Big	Laptop
10	**9**	English	Chrome	Small	Phone

Figure 2.4. *Example of combinations of variables*

example, the date time class, the equivalence classes would be tested at their limit. To add complexity, February in leap year has one additional day. This is a good example of parameter combinations creating more tests. [1] [4]

Because of complex rules of system, combinations of inputs have effect on output. Decision tables are used for testing combinations of equivalence classes. Boundary values can be included to these tests. Approach to these tests is to identify how input combinations change the output. Combination is also known as multi-variable testing. Pair coverage is usual approach takes less time and effort than testing all combination. Using coverage of three equivalence classes produces more comprehensive tests and amount of tests cases stays manageable.

Figure below shows an example of decision table. Columns show rules and rows show conditions and actions/outputs. Figure 2.4 shows example table of combination testing. The table was created suing an online tool [8]. The example would be some GUI testing, maybe some website. Column and row variables are inserted to create combinations. Using this tool to maximize the benefits of test output and minimize used test time. This example used option 'generate pairwise' to limit the number of combinations. The tool can create a table with all possible combinations. With the table's variables there would be 54 different variants. With the same online tool combination options can be changed by banning some combination or changing some variable to must be in every combination.

2.4 Automated testing

Initially, tests are executed manually by testers. It is possible to create software that can execute tests and gather reports instead of testers. This is referenced as test automation. Sometimes test automation is so huge software development, it is a project itself. Now days test automation is crucial to ensure the quality of software components. Before

automating tests, it is vital to understand the possibilities and limitations in automation. Without the review of these the whole project can fail. When automation is started, there are problems due to its expectations and limitations. There are general directions on how to make them realistic. Sometimes a project has too much hope for automation, and it's thought to be a answer to all problems when it's not. This section of chapter also includes how to approach towards to different testing levels, and how their implementation differs from each other. How to choose right tool for right tasks, and how to react to changes in the future. This means test maintainability. [7] [9]

There are a few common reasons why developers choose to automate their tests. There are four basic things test automation can bring, which are increased test coverage, reducing time for regression tests, reduction of manual tests and speeding up the development process. Test coverage is measured by used test data. Adding combinations for test data increases test coverage, but there are too many possible combinations to test manually. Test automation can do these tests repetitive and continuously with almost unlimited set of data. It is also easier to verify product quality when computers check the test results. Regression tests are for verifying the functionalities of new release of the system. These tests are executed before the new version is deployed. Test cases are same for each release, unless the requirements change, so automating those tests works on every release. Reducing time used for testing is major reason for automation and automating regression tests is one of the most common subjects of test automation. Manual tests are often repetitive, and use of test scripts can release testers from them. Testers can focus more on tests that are not or could not be automated. This saved time can also be used to improve old tests. Development process can be accelerated when test execution and result handling are automated. These actions save developer's time and that can be used for other parts of the project. For example, a technical debt is a serious problem when there's no time to correct it. Tests can be executed quickly and outside of working hours. If nothing else is automated in testing, the unit tests are advisable to automate. This will give bug and error reporting from the beginning of the project. Bugs are quickly discovered and reported without much of human input. Reducing used time in long run saves resources. [7] [9] [3]

When there are positive aspects to automation, there also negative ones. Problems and limitations can arise in any part of the project, and it is important to recognize them so they can be addressed. Limitations of automation refer to difficulties to turn manual tests to automated. This is the first problem of automation. Automated tests reinforce manual tests, not replace the and bad test planning creates bad automated tests. System and component requirement changes invalidates old tests. For this reason, maintaining coherent tests take time and effort. It's not always easy to choose how much is invested to automation and end results are not always clear. [7]

There are various reasons to not to implement test automation. Problems may multiply

if the tests automation is started in the middle of project. Implementing test automation means it must be factored in the project from the start. Next is a list of a few reasons. Second reason is the difficult to known what has been coded. This extends to the system and its tests. It is sometimes difficult to verify that the code and tests follow the requirements. Code changes may change the behaviour of automated tests and this way the tests may not be correct. Another reason is the difficulty to verify the test results, meaning the data from automation test reports may be incorrect. Implementation not following requirements, poor quality of test planning and bugs in test scripts may cause false positive passes to tests. Bugs in tests may prevent failure reporting. Lack of human insight is a follow-up from previous problem. With insight of tests, obvious failures of buggy or incoherent tests can be captured. This means the person who examines these tests must have good understanding of scripts and requirements. The problem with insight means that the automated tests must be checked manually.

Technical limitations refers to how limitations of testing tools can impact automation. Some tools have limitations in what kind of environment they can be used. Some requirements, like speed and accuracy, might be too troublesome for some. Some tests' complexities might be too much to handle. These kinds of tests can be generated when multiple unrelated components interact with each other. For example, some aspects of GUI testing cannot be automated. The UI's appearance and intuition to humans are subjective and must be tested manually. Script maintenance happens when the system is updated. Changes in any part of the system means the tests have to be updated too. This includes reviewing the requirements, updating, and creating new designs and implementing them. Maintenance of tests is important because the system updates invalidate some old tests. This is a problem that can cause insecurity to some developers. Last problem with test automation is with high initial investment. Using automation tool is not just one task among the others in the project. Test automation is a development project itself inside the project. Automation test plan could fall through if this problem is not addressed. All these problems share same difficulty, which is how much resources should be used to get good enough results. [7] [9] [3]

Test automation can divide the tests into three categories, which are unit, service, and UI testing. Figure 2.5 presents test automation pyramid. The unit and component testing are in the lowest level and it supports other levels. This means that to service and UI tests to be successful, the unit tests must done first. Unit test level should have approximately 90 percent of automated tests, service level maximum of 15 percent and UI level 5 percent. Service level of testing includes acceptance tests at API level. [10]

Automation strategy is the guideline test automation. It covers the whole software development from project analysis to system monitoring and maintenance. Automation strategy presented here is generally followed in any software projects. The strategy is divided to four different phases. These are feasibility analysis, plan, implementation, and monitor.

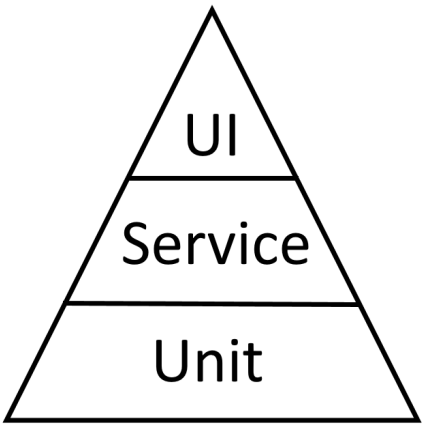

Figure 2.5. *Test automation pyramid*

These phases are somewhat similar to testing process. Plan and implementation phases follow similar rules as creating manual tests. First phase, the feasibility analysis, is about resources used in automation from the business standpoint. Automation is an expensive activity, it uses plenty of money, work power, and time. For this reason the first act is to determinate if automation could bring benefits to the client. Wasting resources is meant to be stopped right front the beginning. The phase is big enough to be divided to five parts. First part is about business need analysis where benefits of automation are brought to the project. Justifications of automating tests are gathered. Second part is ROI calculations where the possible return of investments is calculated. Approaching the automation form perspective of used and gained money is the main point of this part. [7]

Third part is feasibility reporting which includes gathering the pros and cons of automation. After this they are weighed to decide the level automation. Report and ROI calculations are shared with the client to help them to understand the automation effort and scope. With these the client can make decisions more realistically. Fourth part is tool selection. Here available tools in market are reviewed to see if they fulfil project requirements. Also cost for tools' licenses are considered. Last part of feasibility analysis is proof of concept. It is done for a complex sample scenario. Results from this scenario is to get a good understanding of the test environment, tools, and requirements. The part is done also to see the limits of planned tools. In this phase it is possible to choose different tool or set of tools without causing much trouble to testing. Most important part of this phase because it shows the benefits of automation to developer and the client. [7] [2]

After the feasibility check comes planning for the tests. This phase is started only if it's decided to use automation in testing. Automation framework and necessary technical skills are defined and identified. Functionalities of tests are specified and prioritized. Requirements for tests and test data are defined. Next phase of automation strategy is implementation, and it follows quite same steps as project's software implementation.

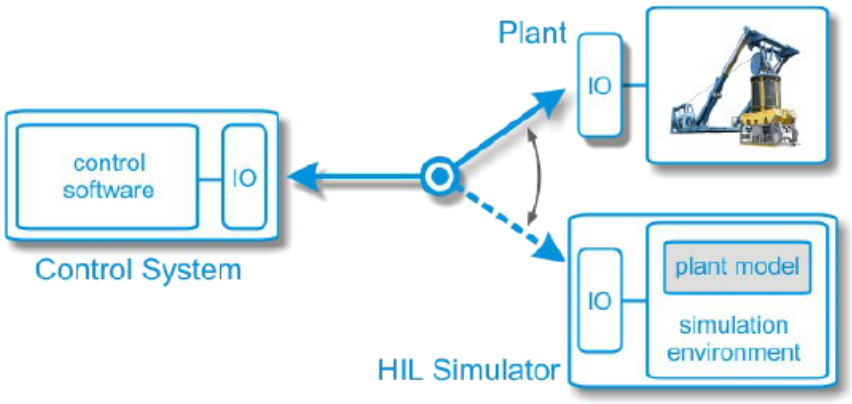

Figure 2.6. *Connecting the control system with simulated plant [11]*

These to follow hand in hand with each other, having an impact on with each other. The last phase, monitoring, is evaluating the impact of test automation to production. The phase includes tests' maintainability, a problem that was mentioned earlier and needs to be considered at the feasibility analysis. Monitor phase gives feedback that is used in planning phase. With the feedback the tests are updated to meet new changes. This creates a feedback loop with plan, implement, and monitor phases inside it. [7] [2]

Automation can be divided to three main approaches. First approach method is Capture and Replay. It is the most primitive and the most inefficient method to execute tests. Manually done test is recorded and later replayed. The record can be sequence of test data or user inputs. The records are always linear and have to be reconstructed when tests are updated, because the variables is the recording scripts are hard coded. Second approach is data-driven tests case presentation. The main difference from the last method is the test data is replaced by variables and they are assigned from specified data tables. This allows test executions with multiple combination of inputs. Combination tables are used in this approach. Keyword-driven test case presentation is the last approach method. The test is divided to smaller steps that can be combined to create complex test sequences. Tests data is replaced with variables like in data-driven approach. It is easier to change tests, because only singular smaller tasks need to be updated, not whole test script. [10]

2.5 Hardware-in-the-Loop

Hardware-in-the-Loop (HIL) simulation is used for the developing and testing control systems that are part of bigger system. A physical part of the system is replaced by a simulation. A simulation can have some physical parts, or it can be fully virtualized. Figure 2.6 shows the basic idea behind IHE. The control system is meant to be tested. It is disconnected from real plant and is connected to simulator. With that switch the plant can be used normally with other control system while the original one is being tested in a controlled environment. [11]

Testing a real system can be time consuming or dangerous for itself or surroundings. To prevent these, a system's behaviour is simulated before tests on physical system is deployed. Failing a test in simulation is much safer option than with real one, and simulation doesn't destroy any physical parts. HIL can be used in all stages of design. With a simulation it's easier to gather test results. tests can be automated and be run continuously. All this combined saves resources. [11]

3. DESCRIPTION OF TEST BENCH AND EQUIPMENT UNDER TEST

Due the huge number of products the company develops, this thesis is limited to three products: 227VM actuator used in air condition pipes called Ulsa, Fico Pro control system for fire dampers and ultrasound sensor system. Every one of this equipment, their use, technology, and tests are explained in this chapter.

Before analysing existing tests it's critical to get to know used technologies and their behaviour. There's lots of physical equipment that are handled, so it makes reviewing the equipment important. This chapter is divided into four parts. The first part explains Modbus protocol that is used to communicate between devices. The reason for explaining Modbus thoroughly is because most of the equipment used in this thesis utilize Modbus protocol and it's also used in testing. In company's products the use of Modbus is widespread.

Second part of this chapter is about test laboratory, a place where equipment is tested. This part is here to help to understand equipment testing before explaining the tests later in this chapter. The last part focuses on describing thesis' equipment and their tests. This section first explains how to divide the equipment into three different levels.

3.1 Modbus

Modbus is serial communication protocol where master-slave architecture, also known as parent-child, is used to communicate between devices. It is used mainly in industrial manufacturing environment and there are currently over 7 million Modbus devices in North America and Europe. Modbus' wide distribution can be explained by its adaptability for different purposes. Modbus sends and receives data via a serial bus line. Modbus is developed by Modicon in 1979. Modbus Organization, an independent, member based, non-profit organization. [12] [13]

There can be only one master device and maximum 247 slave devices in one Modbus serial communication bus. Slave devices don't communicate with each other. The master device sends a request to singular or multiple slave devices, and they respond. Slave nodes won't transmit data without master's request. Only one request can be handled

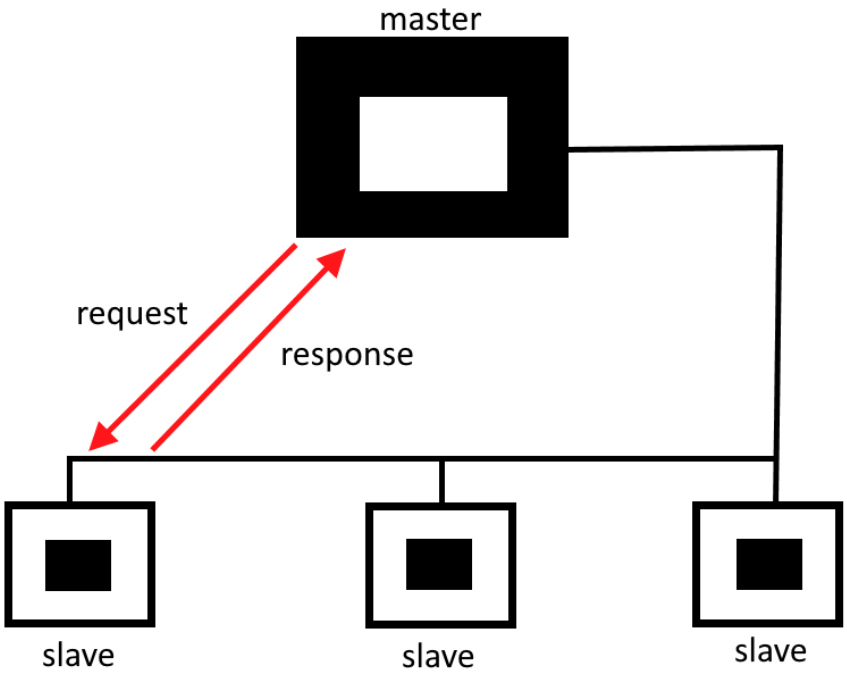

Figure 3.1. *Unicast mode*

at the same time. For each request there can be only one response sent back. A request can be about controlling slave devices or asking slave's information. This could be measurement data or the slave's state. Each slave has a unique address code so that the request can be sent to specific device. There are two request modes, unicast and broadcast. In unicast mode the master sends a request to individual slave device. The slave response according to the request. Figure 3.1 shows master-slave communication between the devices in unicast mode. Master device sends a request to a singular slave device, and it sends back a response. [14]

Figure 3.2 shows the same setup as in figure 3.1, but this time the communication is done in broadcast mode. In broadcast mode the master sends a request to all slave devices, but slaves wont's respond back. The request is can only be a writing function because there's no response.

Modbus uses two different data types: coils and registers. Coils use a single bit, and they are used for truth values. When coils present input values, they contain status of physical discrete data. It is same case with output values; they have the state of physical discrete output signal. Registers are 16-bit unsigned register data. This means the range of data is limited to positive integers, from 0 to 65535. Registers are divided to input and holding registers. Input registers report external input's state. Holding registers are used

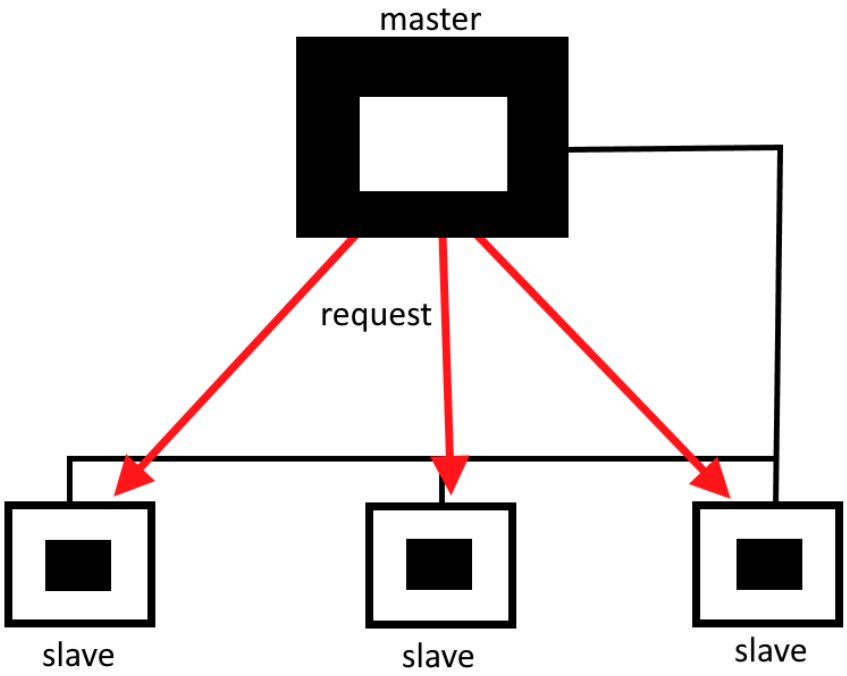

Figure 3.2. *Broadcast mode*

Data Type	Raw Data Type	Type Of	Comment
Discrete Input	Single bit	Read-Only	Its value can be provided by an I/O system
Coil	Single bit	Read-Write	Its value can be altered by a client application program
Input Register	16-bit word	Read-Only	Its value can be provided by an I/O system
Holding Register	16-bit word	Read-Write	Its value can be altered by a client application program

Figure 3.3. *Modbus data types [16]*

to hold device's data storage. All these data types can be found in figure 3.3. The figure holds basic information for each data type. Partially on the simplicity of message data the Modbus RTU is light communication protocol. [14] [15]

Modbus message includes slave device's id, function code, data and error checking field. Function code tells slave machine what kind of action to perform. These can be either read or write commands. With read command the slave sends its state information to the master. Write command manipulates slave's state and behaviour. Data field contains request and response parameters. Error checking field is based on a Cyclical Redundancy Checking (CRC) method performed for the individual characters of the message. Figure 3.4 shows the description RTU message frame. Maximum size of data frame is 256 bytes. [14]

There are three different version of Modbus: ASCII, RTU and TCP/IP. ASCII and RTU are

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low, CRC Hi

Figure 3.4. *Modbus RTU Message frame [14]*

serial transmission modes, they are simple serial protocols. Modbus ASCII mode each 8-bit byte in message is sent as two ASCII characters. In Modbus RTU each 8–bit byte in a message contains two 4–bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII mode for the same baud rate. More about RTU later. Modbus TCP/IP runs on an Ethernet physical layer and Modbus RTU is a serial protocol. TCP/IP is industry standard and it's widely used. It takes the Modbus instruction set and wraps TCP/IP around it. This makes it easy to be added to TCP/IP networks. Modbus TCP/IP follows OSI Network Model and can be used in Ethernet network. [12] [14]

FläktGroup uses Modbus RTU version of the protocol in all of its equipment that use Modbus. One version means there's no collisions between different communication protocols. Some devices, like Fico Pro, can also use Modbus TCP. Modbus RTU is suited for company's need because RTU is lighter than other Modbus versions and to cause harm to the Modbus bus the attacker would have to physically connect their device to the network. Modbus is an ideal protocol for RTU applications where wireless communication is required.

RTU stands for Remote Terminal Unit and it's a binary representation of the data for communication. Communicating with RTU requires that the baud rate is defined, character format and slave address are known. There are no methods for automated recognition of baud rates. All slaves and masters in bus must use same baud rate. A message's frame is separated by a interval of at least 3.5 character times. The entire message frame must be transmitted as a continuous stream of characters. [14]

3.2 Test Laboratory

Hardware needs to be tested to ensure their quality. Sometimes testing needs specific tools and location. Either testing is dangerous or needs specific environment. A test laboratory, or a test lab, is needed. Purpose of test laboratory is a physical place where testing is conducted and it's more of engineering laboratory than scientific research. Practical knowledge is searched there. [17] [18]

Next issues should be factored in when planning a test lab. All this is part of floor plan.

Every issue affect other. In floor plan lab's layout and dimensions are shown as an image. The image uses symbols to describe some objects. First issue is about the size of test lab. Enough space for equipment and testers so they can move around. Extra space could be left for future improvements of tests or new test environments. Should the lab be built from scratch or is there a house or big room for that. Enough lighting means it's easier to execute tests when nothing is hidden is shadows. Windows can provide lighting, but they should be tinted for it brings extra heat to the lab. Layout means positioning the furniture like desks, cabinets, shelves, racks, and equipment to be installed. A good layout plan allows people to move around freely. [17]

Climate control refers to air conditioning and heating. Usually, the lab is enclosed space. Controlling the lab's climate is for well-being of testers and equipment. This way equipment's wear and tear can be minimized and maximize the mileage. Fire safety and prevention includes smoke or fire detectors. For fire controls it's good to have fire extinguishers in various places. Automatic fire suppression equipment is used for electric fire. This can be caused such as mainframes and switches. Fire safety is one of the most important part of floor plan and layout. Equipment needs power to work and Placement of power outlets or connections in the right places. Different electric power outlets need to be taken into account if the lab includes high powered equipment. Backup power could be considered if tests are not allowed to be stopped. Last issue of floor plan is about facilities for staff. Testers and other employees need various building components such as restrooms, stairways, and doors to fully get all productivity from staff. [17]

A lab needs equipment for testing and with inventory check it can fulfil testing needs. Inventory includes software for running and logging tests, hardware to run these applications, consumables like duct tape and printer supplies, office equipment like chairs and printers and tools, for example, electrical testing tools and test management software. Inventory also includes reference materials, which are documented standards and documented testing practises. Risk analysis is part of the inventory. Its purpose is to choose right kind of inventory for lab's purposes. Inventory could be sorter by what is most needed and their cost factors. They are affected by how much equipment is used in testing. For example, if the product is expensive but it is used in various test and its lifespan is years, it might be a good investment. [17]

Next step in test lab is storage of inventory. A bad inventory storage means the equipment is scattered across the lab. This creates situations where devices used in a single test setup need to be borrowed around the lab. This can create traffic jam. In floor plan take into account if in future more tools are acquired. New equipment need space to be shelved. Each physical piece in inventory should be given ID. This way it's easier to keep track of their use and wearing. [17] [18]

FläktGroup's test laboratory is used nearly every workday for various products. This in-

cludes equipment used in this thesis. All previously mentioned definitions for test lab are fulfilled. The lab meets all requirements for testing their products. A variety of pipes, fans and measurement devices make huge amount of different ventilation setups possible. The laboratory is spacious enough for many simultaneous testing, even for same type of equipment and tests. Equipment is tested here for durability, like heat resistance and much they can be used before they break, how well they follow safety protocols and accuracy of actuators. With all these testing equipment a controlled environment can be created for testing products.

There are multiple rooms for different tests at same location at Akaa. There are couple separate rooms for testing ventilation equipment, a heat chamber for testing heat resistances and custom built testing wall for fire safety system. Equipment and tools used in testing are explained later in this chapter.

3.3 Differences of devices

Every type of company equipment goes through system and acceptance testing whenever there's software updates or new patch of produced hardware. The equipment is divided by who developed them. This affects how they are tested. There are three different levels. First equipment level contains those devices that are developed by other than FläktGroup. Device's testing consists of fully black box testing. This includes system and acceptance testing.

Second equipment level contains those devices that are developed in collaboration with Fläktgroup and other companies. This makes testing a mix of white box and black box testing, making unit and integration testing possible. Third equipment level contains those devices that are developed fully by FläktGroup. The level focuses on white box testing. Testing approach depends on who created the equipment. White box testing is excluded if software is developed by other company. Next is a short list of equipment used in this thesis divided to their equipment levels:

- Level 1: Ultra Ulsa, specially actuator 227VM
- Level 2: Fico Pro
- Level 3: Ultrasound air flow measurement device

Documentation for test cases exists but not for equipment setup or detailed instructions for testing. The instructions included what functions are tested but much about their setup. The focus was on interviewing testers and monitoring the tests while they were executed. The interviewing and monitoring happened mostly side by side. The tester explained the tests while executing them. Sometimes only explaining tests and taking notes. More information was asked about some steps when more clarification was needed. Interviewing and monitoring happened multiple times at the laboratories. Each level of equipment had

Figure 3.5. *Ultra Ulsa and Ulda [19]*

different testers. Level 1 equipment testing was mostly monitoring the testing. Interviewing was done while executing tests. Level 2 equipment testing was mostly interviewing the tester. The tester showed testing equipment and how tests are executed. The basics of test cases were presented and how to document test results. Level 3 equipment testing there was no documentation for tests that are meant to be automated.

3.4 Level 1 equipment

In level one equipment the main focus in on 227VM and Ultra Ulsa which uses it. Modbus is used to monitor and control these devices and are main part of testing. There can be maximum of 128 Ulsas as slave devices in Modbus communication bus. Each device's burden takes one quarter of device in the bus. First, the technical description of Ulsa is given, then the actuator 227VM. Last part of is focusing on describing tests done for these devices. The Ulsa is part of this level because it is main component when testing 227VM.

Variable air volume (VAV) dampers are used to control air flow inside pipes which are used for temperature and air quality control inside buildings. Non insulated casing version of Ultra is called Ulsa. Insulated version is called Ulda. Pipe, ultrasound device and 227VM are combined in installation. This way the correct positions are easier to accomplish. Figure 3.5 shows Ulsa and Ulda devices. The device mounted on them is 227VM.

The company offers many benefits for these products. Some of these are low noise, measurement doesn't cause pressure loss, causes no additional noise, a wide range for air flow, and large air flow range for different needs. Low pressure drop means less wasted energy used for creating excess airflow. The company offers nine different sizes for Ultra products, diameter of pipes starting from 100 mm to 630 mm. Operation range changes according to the sizes. Air flow velocities between 0.5 and 15 m/s are for sizes 100-315 mm and 0,5-13 m/s are for sizes 100-630 mm. Damper's bearing part of Ulsa made of nylon and its shaft is mounted in nylon headings. [20]

The damper Ultra is used for controlling constant and changing air flows. The functionality

Figure 3.6. *227VMZ compact controller [21]*

of the damper includes forced shut of flow, upper and lower limits for velocities. These can be accomplished by the ultrasound sensor and 227VM's damper blade. [20]

227VM is a controller unit for handling air flow inside pipes. It is part of Optivent Ultra products. The device includes real time air flow monitoring, pressure sensor and user interface. Figure 3.6 shows 227VM device, which includes sensors, actuator, and user interface. The interface consists of small display and two red selectors. 227VM controls a blade located inside the pipe to control air current. The actuator moves the damper blade to control air flow. With blade position, or angle, zero percent means the valve is fully closed. 100 percent means the valve is at the position of 90 degrees, meaning the air flow can move without barrier. [19]

Figure 3.7 shows front of 227VM device, the user interface. Value selector (1) allows changing values. Changes are displayed. Function selector (2) allows choosing function. The function chosen depends on selector's position. Display (3) shows chosen functions, values, and units. Unit is shown as a dot next to correct one. There are three different units in the display. Air flow has two of these options: litres per second and cubic meters per hour. Third unit shows truth value, used by many functions. Selectors can be used with screwdriver. [22]

The device can be operated via Modbus or with analogue signal. When using Modbus, the device is classified as a slave device. With analogue control the signal is set to between zero and ten voltages. Figure 3.8 shows first three Modbus functions for 227VM. First column is function's index number. Second column is the name of the function. Address column is function's Modbus address to the device. Data type and value range

USER INTERFACE

Figure 3.7. *227VM User interface [22]*

OPERATION VALUES

Number	Name	Address	Data type	Value range	Read/write	Description	Default
1	Setpoint	0	WORD	0...10 000	r/w	Shows/sets Setpoint [%] 0 ...100.00% matches on Vmin and Vmax 0% = Vmin 100% = Vmax The register is read only, if register 122 = 0 or 3	-
2	Override control	1	WORD	0...4	r/w	Sets Override '0' No override '1' open '2' close '3' Vmin '4' Vmax '5' Vbetween	0
3	Command	2	WORD	3	r/w	'3' Starts calibration (sw 2.77 and newer) Do not write any other values	-

Figure 3.8. *Some Modbus functions for 227VM [22]*

tells what kind of data is used in the function. Read/write column tells if it's read and/or write function. Description tells the functionality and how each value and value ranges affect to it. Some functions have default values.

Testing these devices is considered black box testing because only system's behaviour is known. This influences testing approach. In V-model tests done for Ulsa are system tests. Ulsa is a ready product that still needs testing. Tests at this level also follow the principles of HIL, as the tests simulate as if the equipment were installed in a house. The company uses and codes with LabVIEW. It is system-design platform and development

environment. It is mostly visual programming language meaning instead of writing code the program is constructed using blocks and connections between them [23]. Some tests use LabVIEW for test setups and test results logging.

There are plenty of physical configurations for testing this equipment. Combining different angles, distances, and connections between physical components like pipes, actuators and sensors creates test cases. Reason for this many configurations is that clients have huge variety of buildings that have different needs for ventilation. There are approximately 400 different configurations for Ulsa alone.

Modbus Poll, a program that virtualizes Modbus master, is used in testing. This makes a computer a master device in bus, and it's easy to control the communications. The program is meant for testing and developing Modbus devices. The program supports some Modbus functions, which makes it enough for testing. It's easy to configure slave devices: specify slave's identification code, function, address, size, and poll rate. Can monitor several slaves and data areas at the same time.

Two different programs are used for testing. First program, Flow.vi, controls air flow directed to pipes. This program was created using LabVIEW. Second program, Modbus Poll, handles ULSA's actuator and sensors. The tester uses this program to manipulate actuator and collect measurement data. Combination testing is used: testing all possible cases. This creates over 400 different combinations. Each test case has three different airflow speeds and three different blade positions. Combining these creates nine measurements in one test case. This means all possible scenarios are tested, creating huge amount of test cases. The company sees this is acceptable for the buildings' fire safety depends on this equipment. Three different fans to control the air flow. T-links of pipes and multiple air flows are tested.

Ulsa's testing consists of average value of measured air flow in pipes. Next paragraph gives a description of manually done testing to Ulsa. First, configure physical components as they are described in test plan. Second step is to change the air flow and blade position. For each measurement the speed and position are given as test input. Airflow speed is changed manually, and input doesn't always correlate with real air flow speed, so adjusting is required. Adjusting takes time because fans change the peed slowly. Measurement can start when airflow speed and blade position have become steady. Average value of air flow inside pipe is calculated from a thirty second sequence. Tester calculates average value of speed and enters this value to second program for logging. Blade position is also logged. Fourth step is to create report using the first program that creates excel file. Testers adds date and their name to the file. The last step is done when all speed and position combinations have been logged. T-branch connections of pipes need additional calculations because of complex air flow the branch creates. Testers use different excel file for these measurements.

Function	Tested activity	Measured situations	Date tested	Tester	OK / NO	Observed malfunctior
Commissioning	Reference drive	Size 200, size 125 and size 160	28.3.2018	JTM	OK	
	Reset + Calibration after start-up	Size 200, size 125 and size 160	28.3.2018	JTM	OK	If you write to some registers during cal, cal text will stay on display also after cal is finished
	Actuator size matches to size sensor is reporting	Size 200, size 125 and size 160	28.3.2018	JTM	OK	
	Change of installation parameter, both actuator and sensor	Size 200	27.3.2018	JTM	OK	
	Vnom table (l/s and m^3/h)	Size 200, size 125 and size 160	27.3.2018	JTM	OK	
	Change of unit parameter, both actuator and sensor	Size 200, Size 125 in air flow	28.3.2018	JTM	OK	
	Changing Vmin, Vbtw and Vmax values	Size 200, Size 125 in air flow	28.3.2018	JTM	OK	Changing value removes override set by MB
Forced control via MB	VAV regulating to Open, Vmin, Vmax, Closed, 2-10V	Size 125 in air flow, MB contol, analog control	9.4.2018	JTM	OK	
	VAV regulating to Open, Vmin, Vmax, Closed, 0-10V	Size 125 in air flow, MB contol, analog control	3.4.2018	JTM	OK	
Forced control from actuator	VAV regulating to Open, Vmin, Vmax, Closed, On (keep current position)	Size 125 in air flow, analog control	9.4.2018	JTM	OK	
	VAV regulating to Open, Vmin, Vmax , 0 - 10V	Size 125 in air flow, MB contol	9.4.2018	JTM	OK	

Figure 3.9. *Part of test plan for other 227VM tests*

These tests' output data is calculated from measurements of the ultrasound device. There are own tests for this device to examine its quality. Tests assume the device works properly. Ultrasound device's technical and testing review are explained in level three equipment.

There's also tests for 227VM only, without pipe of ultrasound device. Figure 3.9 shows Excel file containing 227VM testing plan. The figure shows couple fist test cases. One test case can have multiple activities, and these can have multiple measurements. This means one functionality is tested multiple times with different adjustments. First column from the left shows the functionality to be tested, a created test case. Second column shows activities of function. Each function can be divided to multiple activities. Third column contains different physical setups for each tested activity. Other columns show test date, tester's name, test's success, found faults and other comments. Information about software and hardware versions are also included.

3.5 Level 2 equipment

ISYteq FICO is a product family that is used to control and monitor fire safety dampers [24]. Fico is connected to multiple slave devices in Modbus bus. Slave devices are connected to multiple fire dampers. The slaves and dampers are connected to a variety of alarm devices, like smoke alarms and alarm buttons. There are two to three alarm buttons for each slave device.

This thesis focuses on Fico Pro. It is used in larger building applications. Figure 3.10 shows front of Fico Pro. It has graphical user interface with a 7-inch touchscreen. Connections for automation systems for buildings. It can use both Modbus RTU and TCP for communicating between the Fico and slave devices. Can monitor up to 256 fire damper actuators and can connect to Ethernet.

Fire shutter control and monitoring system Fico is developed in collaboration with another company. Software versions 1.8.1 of this equipment is examined. Version 1.9.10 is

Figure 3.10. *Fico Pro damber control and monitoring system [25]*

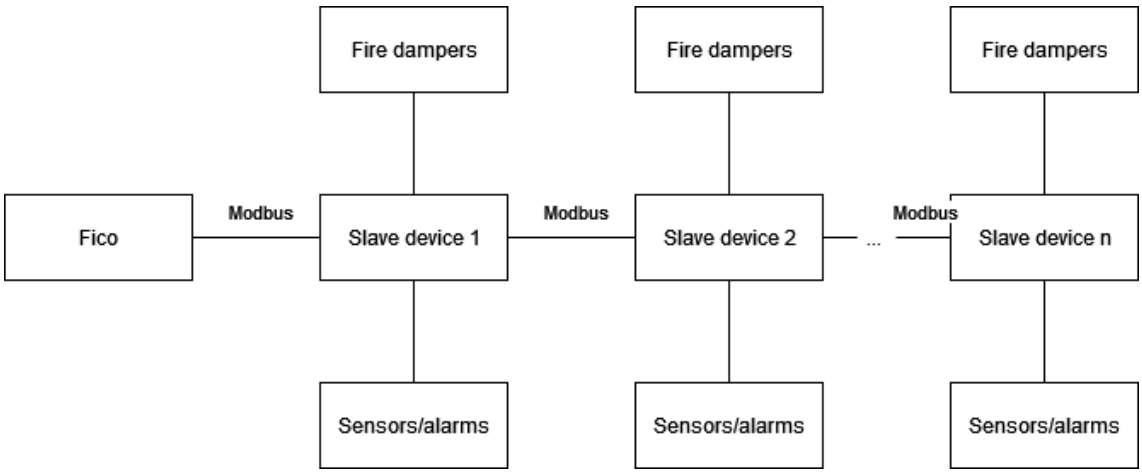

Figure 3.11. *Fico connected to slave and fire safety devices*

newest version currently used by clients. By default all slave devices are listed in default list in Fico. This requires all necessary devices are found in Modbus network. Rooms can be created in Fico and user can choose devices for each room.

Figure 3.11 shows the architecture for connecting devices in a Fico-controlled system. This setup is used for both houses and testing. In the figure, blocks present devices. There can be multiple sensors, fire dampers, and slave devices controlled by a single Fico. Connections between blocks show what devices are connected to each other. Text above connections shows how this is achieved. User monitors and controls the system from Fico's GUI.

Testing Fico Pro includes black box and white box testing. This makes level two equipment

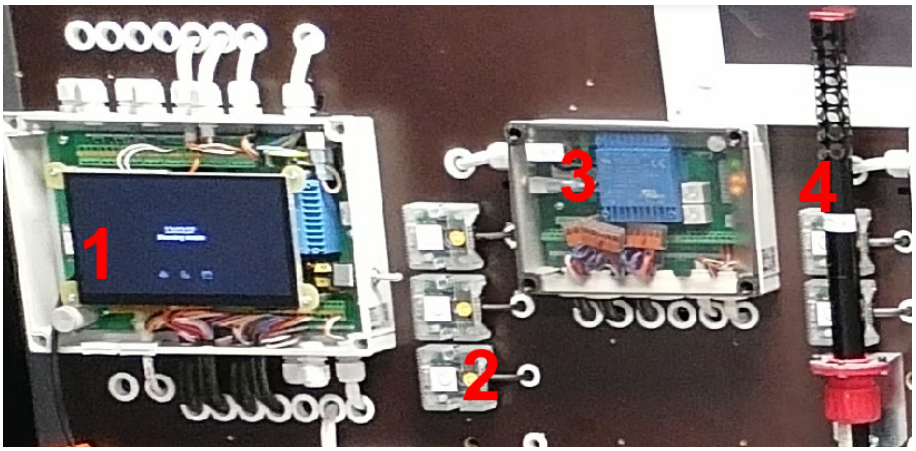

Figure 3.12. *A piece of Fico test setup*

testing a mix of levels one and three. In V-model tests done for Fico are system tests. The device is a ready product that still needs testing. Tests at this level also follow the principles of HIL, as the tests simulate as if the equipment were installed in a house. Testing fire situations and how devices response to them are crucial for for this product. In company's laboratory the testing focuses on black box testing because none or very little is known about the inner workings at the company location. Reported bugs are mostly fixed but the reasons for such are not explained. All system inputs and outputs are done manually. Tester reads outputs from GUI of Fico and slave machines readouts of lights. Modbus architecture is important when testing Fico. Testing Fico controller as master device and with dozens of slave devices. Each slave device has couple sensors. It's easy but time consuming to build test case scenarios. Easy on-off configuration on slave devices. Simulation and fire alarms easy to trigger. All this is done manually.

Figure 3.12 shows part of the test setup. Important devices of the setup have been highlighted with red colour. Devices attached to one wall. Number 1 is Fico unit and it's located at the upper left side of the wall. Number 3 is a slave device that controls alarm devices and fire dampers. There is total 63 slaves in the setup, with normally less than half are used. Number three is button that presents alarm. Number four is smoke alarm. There are multiple alarms, both kinds, in the setup. Fire dampers are located behind the wall. Testers can see states of slave devices in their lights.

Before starting the tests, the Fico device's software is updated. It is connected to internet and after updating factory reset is used. After that the testing can begin. Test plan for Fico Pro covers most scenarios where Fico is used. Focus is on the most likely use cases that clients encounter. Executing these tests takes approximately one workday.

Some possible test cases are not tested due to them being very hard to simulate or setup, very unlikely in customers' end, or the time used for them is seen as waste of time. For example, complex collections of slave devices are rare and because of similarities of house of floor plans the customers create same type of collections. For example, how

TEST CASE ID	TEST SCENARIO	TEST CASE	PRE-CONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	POST CONDITION	ACTUAL RESULT	STATUS (PASS/ FAIL)
TC_IOWIZ_001	Verify settings made with I/O wizard from quick start	Verify input setting for default group	admin login & home view	1. Tab "Groups" tile 2. Tab "default" group edit icon 3. Tab "Configure" 4. Tab "Input" 5. Verify that the X1 (External alarm) & X2 (External interlock) are assigned		X1 (External alarm) & X2 (External interlock) are assigned for defaut group inputs			Pass
TC_IOWIZ_002	Verify settings made with I/O wizard from quick start	Verify output setting for default group	admin login & home view	1. Tab "Groups" tile 2. Tab "default" group edit icon 3. Tab "Configure" 4. Tab "Output" 5. Verify that the REL1 (Fire alarm) & REL2 (Service alarm) are assigned		REL1 (Fire alarm) & REL2 (Service alarm) are assigned for defaut group outputs			Pass
TC_DEFFAN_001	Verify default fan setup	Verify default fan settings	Admin login & home view	1. Tab "Fan" tile 2. Tab "Fan 1" edit icon 3. Verify that the I/O assigned for Fan1 is Controller - REL3		Fan 1 is configured for Controller relay output 3 (REL3) by default			Pass
TC_DEFFAN_002	Verify default fan setup	Verify default fan assignment	Admin login & home view	1. Tab "Groups" tile 2. Tab "default" group edit icon 3. Tab "Fans" 4. Verify that the Fan 1 is assigned to default group below "Fans in group" column		Fan 1 is assigned to the default group by default			Pass
TC_AUTODETECT_001	Verify autodetect functionality	Verify slave unit detection	FICO PRO factory reset and restarted. All the slave units powered up and peripherals detected	1. Finnish Quick start 2. Wait for approx. 8 minutes to FICO PRO detect all the slave units 3. Verify that Devices count is 64 4. Tab "Groups" tile 5. Verify that the damper count is 120 and smoke detector count is 11		All the connected slave units, fire damper and smoke detectors are detected			Pass

Figure 3.13. *Part of test plan for Fico Pro*

system behaves when electricity is cut off during fire alarm. Normally customers don't create complex room and device collections. So there's no need to test too complex combinations. This reduces tests cases. More device and room complexity means more test cases. This is exponential and automation would help to reduce used time and create combinations.

For testing Fico Pro some test cases are described in figure 3.13. Tester log the results of each test scenario. This includes pass/fail status and behaviour or output if they differ from excepted results.

3.6 Level 3 equipment

Ultrasound airflow measurement sensor used in Optivent Ultra products. It is integrated inside the product. The sensor transmits and receives ultrasound to measure air flow inside pipe. It gives accurate measurements to in both high and low air speeds. There are no physical sensors inside the pipe, so the ultrasound device doesn't disturb air flow, cause turbulence or voice. It also doesn't collect dust. This makes the expected lifes of the product longer than the device were inside the pipe. [26] [20]

Due to sensor's small size, the Ulsa is easy to install in various house environments. The device is connected to 227VM that analyzes air speed to create high precision data. Figure 3.14 shows ultrasound device when it's integrated to Ulsa pipe. Black plastic cover protecs electronics and sensors. Cables are connected to 227VM where it sends data. [26]

Ultrasound device's software is fully developed by the company. The software was written in C language and developed in Atmel Studio. In V-model tests done for ultrasound device are system tests. The device is a ready product that still needs testing. The basic functionalities are tested in this level. This means it is black box testing. The device is tested separately from Ulsa. These tests are not explained in detail because the focus is on white box testing. The company's laboratory at Akaa only focuses on black box testing

Figure 3.14. *Ultrasound device integrated to Ultra Ulsa*

with this device.

4. AUTOMATED TESTS

Now that equipment and their tests are explained, it's time to automate the tests. Tests are mostly kept in same level of testing as before. In this chapter automated tests are designed and developed. In the first part of this chapter the manually done tests are analysed to find tests to automate. Criteria for them are explained. In second part the automated tests are designed and implemented. Identifying possible software testing tools is part of this phase. Third part has a description of chosen tools, main algorithms, and snippets of the most important codes.

The author of this thesis has discussed the need for automated tests with testers who have previously done the testing manually. Combining this gathered information with the information of tests that could be automated it was easier to prioritize the tests. This knowledge helps to understand the possibilities of automation and what it can produce to equipment testing.

This chapter follows phases of automation strategy. They begin after analysing old tests. Test requirements are gathered from analysing manually done tests. First phase is feasibility phase. Primary the intent is to use resources of the company that it already possesses. With software new options may be considered, but license costs must be kept in mind. After the analyse the rest of the chapter focus on plan and implementation phases. Not all planned tests are implemented. These phases use agile development, meaning plans are updated when implementations move forward, and implementation react to plans' changes. The analysing step is revisited when necessary. Finally, after implementation is finished, automation is tested and checked to meet all requirements. This part will be discussed more in detail in chapter 5.

UML diagrams are used to show automated tests' structures, sequences, and activities. Not all tests can be automated due to time limits and technical limitations. And some parts of tests can be automated, not fully removing human component from testing. With Ulsa having physical configurations, like pipe angle and distances, are done by testers. Only inputs from computer can automated using scripting. Data gathering for test reporting can be automated. One way to approach automation is not to think goals of test automation but analysing existing manual tests. Manual tests of equipment are approached from this perspective. Results of newly implemented tests are reported in chapter 5 where

improvements for these tests are discussed.

4.1 Analysis of manually done tests

Because not everything can be automated, prioritization is necessary. Prioritization criteria are chosen so that the company gets good benefits from created tests. Primary criteria for this is how much time can be saved if a test or part of it is automated. Secondary criteria is maintainability of tests. This means how much automated tests have to be modified if test requirements change. Modular design and easily changed interfaces would help this. Trying to prioritize these tests is not always straightforward and can generate different results from first impressions from equipment. Interviewing with testers revealed the willingness of wanting to automate input and output handling. Testers said that input and output handling take plenty of time of their work. It's not possible to automate the test equipment setup because there's a lot of hardware to be set up, like in Ulsa. Analysis for level 3 equipment is passed because there are no white box tests to automate.

Code maintainability is important to think about when automating. Because requirements and technology change the test must be updated according to them. This could be hard if the coder is not familiar with the used tools or programming languages. Someone who maintain the tools and code must fix these tests. Tools and tests are updated, so something breaks, and it must be fixed. use of modular program makes it easier to cut and paste, separate different parts of the program. Modular makes it easier to maintain. Not necessary it scripts are small. Small parts to automate means smaller maintainability. Good to keep in mind what programming languages and tools the company normally uses. Using documentation methods practised by the company. Focus is on documenting changes of input and output handling when automating. This is important because in this thesis not every opportunity of automation will be implemented due to the workload so someone else can do the implementation using these documents.

4.1.1 Level 1 equipment

Manually inserted inputs to two programs and manually done result recording of air flow currents can be automated. LabVIEW is used to control fans because there is already implementation for that. For communication between different programs LabVIEW can use one of its many modules to achieve this. The communication can also be established from Python.

Basic idea of Ulsa testing: Follows same steps like in manual tests. Both follow same basic steps in testing. Adjust fan's air flow velocity using simple feedback controller. When correct velocity has been reached the next step starts. Change actuator's angle and adjust if blade is moving back and forth. Takes measurement and calculations. Repeat

before previous and previous steps. Report to Excel and save it. Problems with the angle of actuator: No precise control and nothing happens when blade is told to move a fraction of degree. To prevent these use positions that are known to cause no problems, move across the goal position and back if little position change are needed, or testing script checks if blade is moving. Basically, the two programs for testing are combined and some testing steps are automated. Thresholds for target values are needed because blade position and fan speed are not always precise.

4.1.2 Level 2 equipment

Test data is minimal and only few cases are used for each test. There's also not much negative testing, only obvious cases are taken into account. Focus is on most obvious errors and use cases. Testing time is limited because there are multiple tests to execute. Using Fico's touchscreen in automated testing is not on focus in this thesis, and previously all tests use the GUI to execute tests. Using the GUI would require testing robot. Multiple test results are read from the screen and giving input to GUI is a slow process. Slave device's states are read from them. Devices' control without Fico is done manually with buttons and switches with control power and connections.

Fico can be run from computer using Tera Term. Tera Term is an open-source free software terminal emulator [27]. It supports TCP/IP and serial port connections. It can run macros, and log both data from script and terminal output. This opens a possibility to automation. Machine uses serial port with speed of 115200 baud. Fico uses Linux as OS; this means user can give Linux commands. Figure 4.1 shows how USB cable is connected to Fico. The connection goes behind the touchscreen.

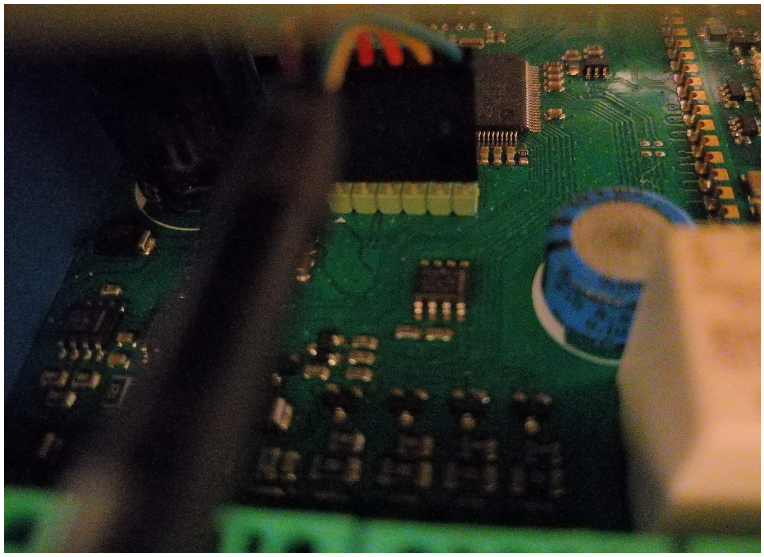

Figure 4.1. *USB connection to Fico Pro*

From Tera Term Fico's program can be accessed. In this program user can give com-

mands without using touchscreen. Further examination revealed that user cannot give many commands to the Fico Pro program. Only commands found were about printing devise's status, reboot application, quit program and toggling machine's online and standby modes.

4.2 Automating tests

Now that all old tests are analysed, the next step is to start designing automated tests. This includes prioritizing tests for each equipment level. This means only some tests are automated in this thesis. Finally automated tests are implemented. Limitations of testing equipment's and equipment's software and hardware limit what can be implemented in this thesis. After last equipment level's design and implementation comes description of test environment. Results and findings of these tests can be found in chapter 5. Not all tests or steps are automated. Automation starts with important test, meaning prioritization. Google Drive is used in version control. After each day when code is modified, all code is compressed and uploaded to the Drive.

Company Modbustools offers other programs for Modbus, like virtualization of slaves and TCP/IP control for .NET. A list below shows supported Modbus functions by Modbus Poll. Left size of each row tells the code number of function and right size tells the name and action of the function.

- 01: Read coil status
- 02: Read input status
- 03: Read holding registers
- 04: Read input registers
- 05: Write single coil
- 06: Write single register
- 15: Write multiple coils
- 16: Write multiple registers
- 17: Report slave ID
- 22: Mask write register
- 23: Read/Write registers
- 43/14: Read device identification

4.2.1 Level 1 equipment

Testing stays in black box testing and at the same level as in manual testing. Tests stay in same level in V-model. Due to the scale of this work, the focus is on Ulsa testing. There are two ways to automate tests: LabVIEW and Python programming languages. Control test execution from one of these, but still use both of them. One controls the other. Some company's LabVIEW code exist that can control the fans. These would need updating so it can be used in automation. This equipment level's automation uses data-driven tests case presentation. Each step can be repeated in different physical configurations.

Modbus Poll offers interface to run scripts with Excel macro and Python. It was decided to use Python, with Python 3.6. From Python library Pywin32, win32com client is used to access Modbus Poll. Newest version of the library is v302. Pywin32 is a library that provides access to many of the Windows APIs from Python. Python script can use win32com client from the library to manage different application in PC, like Modbus Poll, LabVIEW, and Excel. The library has 64-bit build so it can work on old and new OS.

Three different design options can be found in figure 4.2. Each option shows architecture of test and how it's divided to different modules. Stick figure named 'Tester' is a tester who starts test execution. Blocks present modules used in testing. Blocks that are named Fans and Ulsa are used hardware, others are software programs. Connections between blocks show what modules are connected to each other. Text above connections shows how this is achieved. Connection between tester and module shows what program starts test execution and gathers results to tester.

Option A uses both Python and LabVIEW. Former program executes tests and gathers results, latter controls fans. Python program in connected to LabVIEW and Modbus Poll using win32com which is part of Pywin32. Modbus Poll is connected to the equipment using Modbus RTU protocol. LabVIEW controls fans. Python can control and monitor all equipment in the diagram. Using Python happens in two steps. First step is to connect Python to Modbus Poll and equipment. Control and monitor the equipment. Second step is connecting Python to LabVIEW. Use old VI program to control fans and handle test logging. Option A can be modified if LabVIEW cannot be controlled with Python. This means the tester must change fan speeds the old way instead of the automated program.

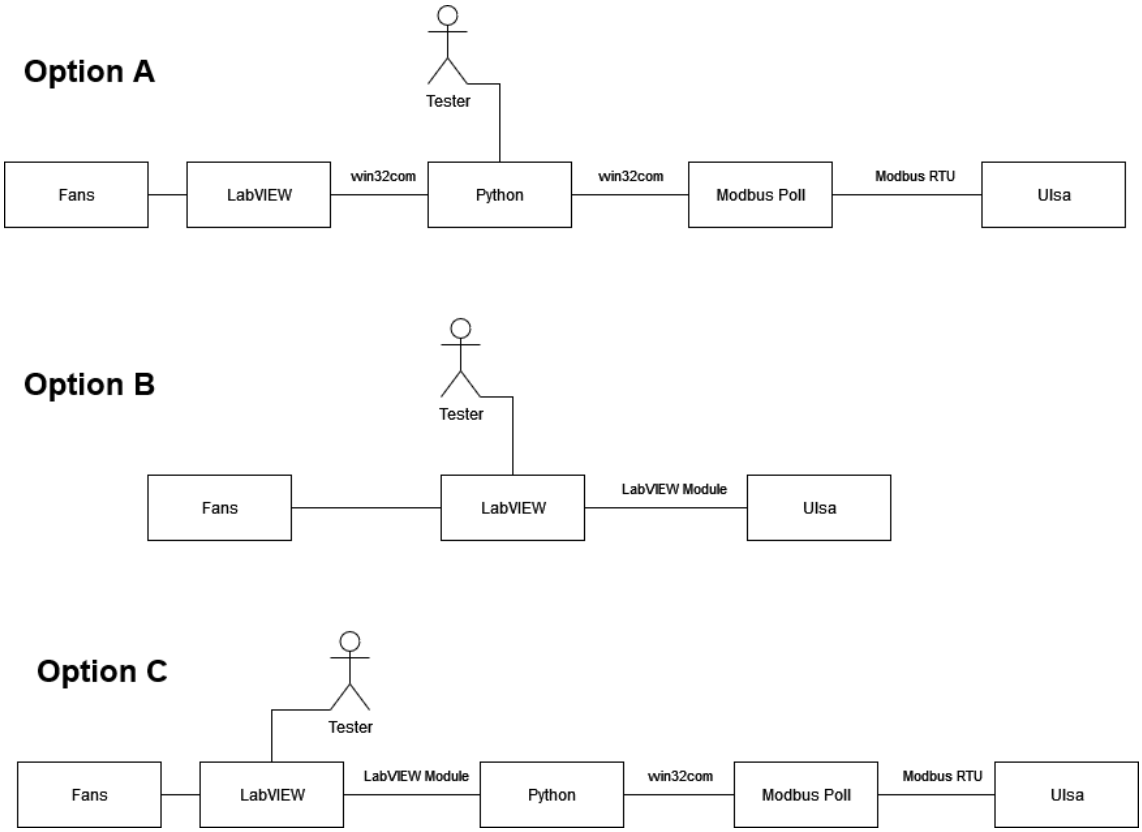

Figure 4.2. *Different options for automating Level 1 testing*

Option B uses only LabVIEW. Tests are implemented in LabVIEW and it is directly connected to equipment using LabVIEW modules. Use code that controls fans to create tests. Might need modification to work properly, because current version of fan control focuses on showing measurements to user. LabVIEW add-on called LabVIEW Data logging and Supervisory Control Module, DSC Module for short. This add-on offers to control Modbus master and slave devices. Three-year licence costs almost five thousand euros. There is a tutorial [28] how to use Modbus in LabVIEW. It offers several options, which DSC module is one of them. There's free alternatives for handling Modbus but usually they are not fully supported or they are not completed or buggy. This means that they could not work when updates happen.

Option C is a combination of options A and B. In this case tests are executed from LabVIEW and results are gathered from there. Python is used to control python programs that are connected to the equipment just like in option A. LabVIEW is connected to Python with a module that can interact with the Python. This option can also use automated tests option found in LabVIEW.

Figure 4.3 shows options on how to control fan speeds. In option 1 the speed control is done in LabVIEW. Because the process takes some time, there's an output that tells if the process is still running. After this output shows that the fan speed has been adjusted correctly, Python gets output that tells speed measurement. In option 2 the feedback

control is done in Python. Bigger speed changes mean longer waiting time. This time could be estimated, and Python can check if the speed is still changing from the output values.

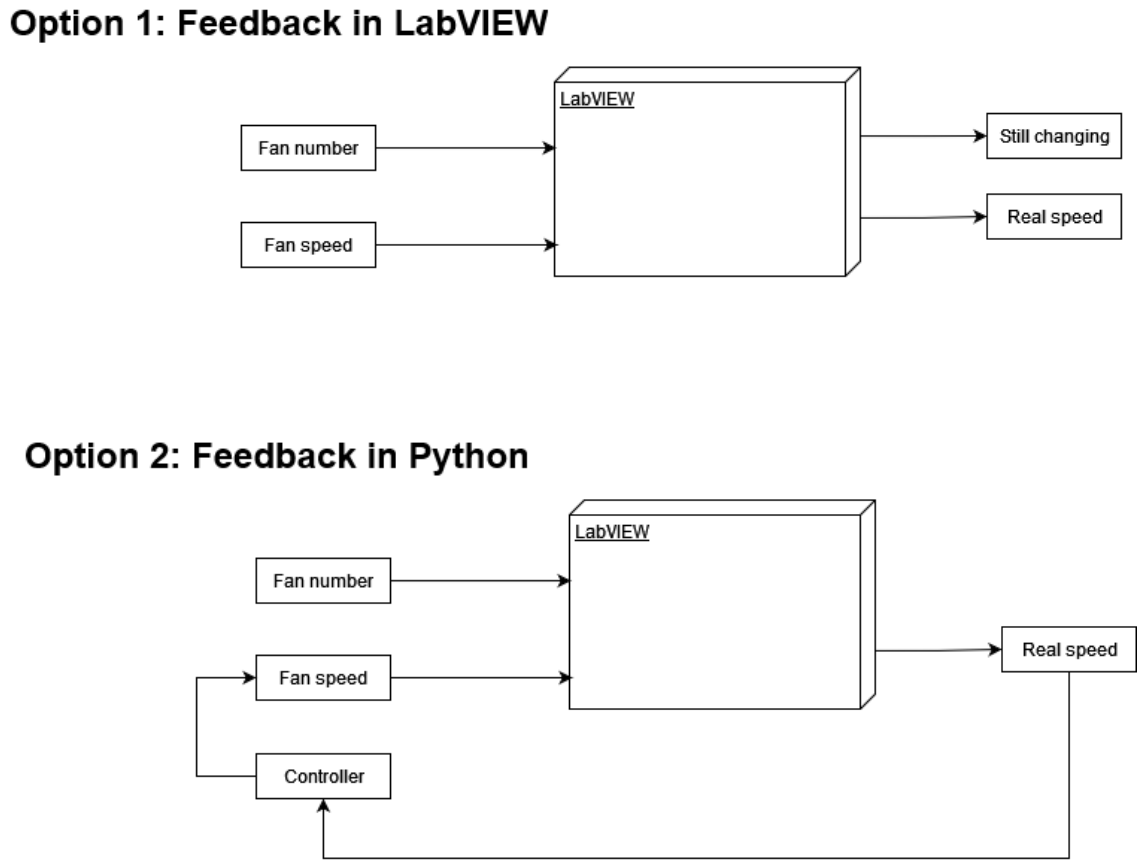

Figure 4.3. *Feedback control options for fan speed*

Next three figures show how same sequence of actions happen in each option. Figure 4.4 shows sequence diagram for option A. In the diagram one fan and blade are moved. This action happens when there are fan speeds left but no positions for current speed. Every data is returned to Python because it logs results from there. First the Python commands LabVIEW to change the fan speed. LabVIEW uses simple feedback loop to get the right speed. After this measured fan speed is returned to Python. Uses option 1 from figure 4.3. Next the blade position is changed. Python monitors moving blade and the loop after blade position reaches a threshold. The blade sometimes sets close to the goal position and the threshold is factored in this. Getting the blade in the absolute goal position is not always possible, and sometimes this makes Ulsa to move the blade back and forth, never stopping. For this reason Python checks if the blade is still moving. If it's moving the new position set near the original goal. This fixes the problem. After all this measurements can be taken.

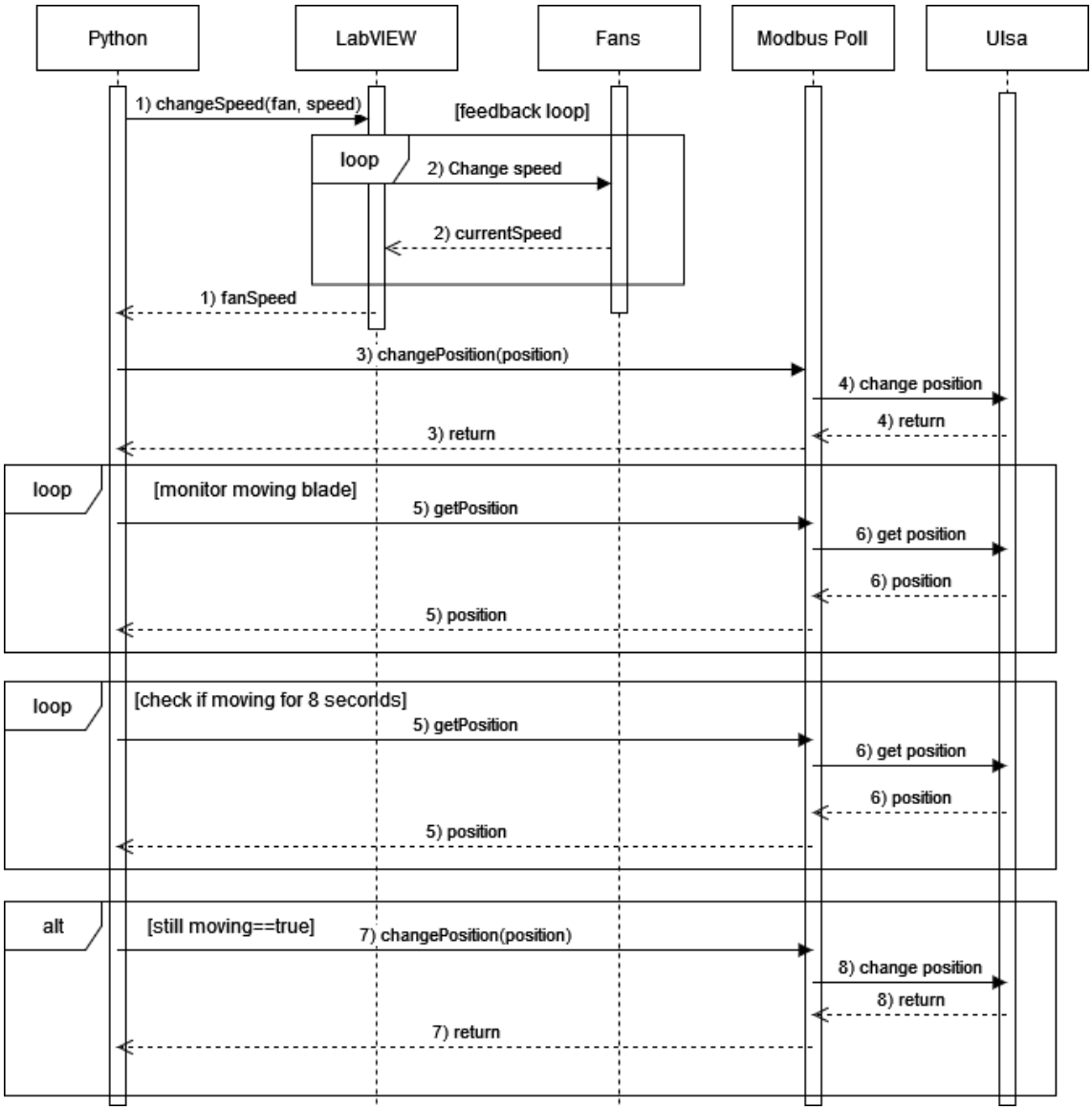

Figure 4.4. *Sequence diagram of Ulsa testing, Option A*

Figure 4.5 shows sequence diagram for option B. The figure has same actions as figure 4.4, but uses LabVIEW as a master module. Option B has simplest structure because LabVIEW takes care of test execution and data logging. This also means code inside LabVIEW becomes more complicated.

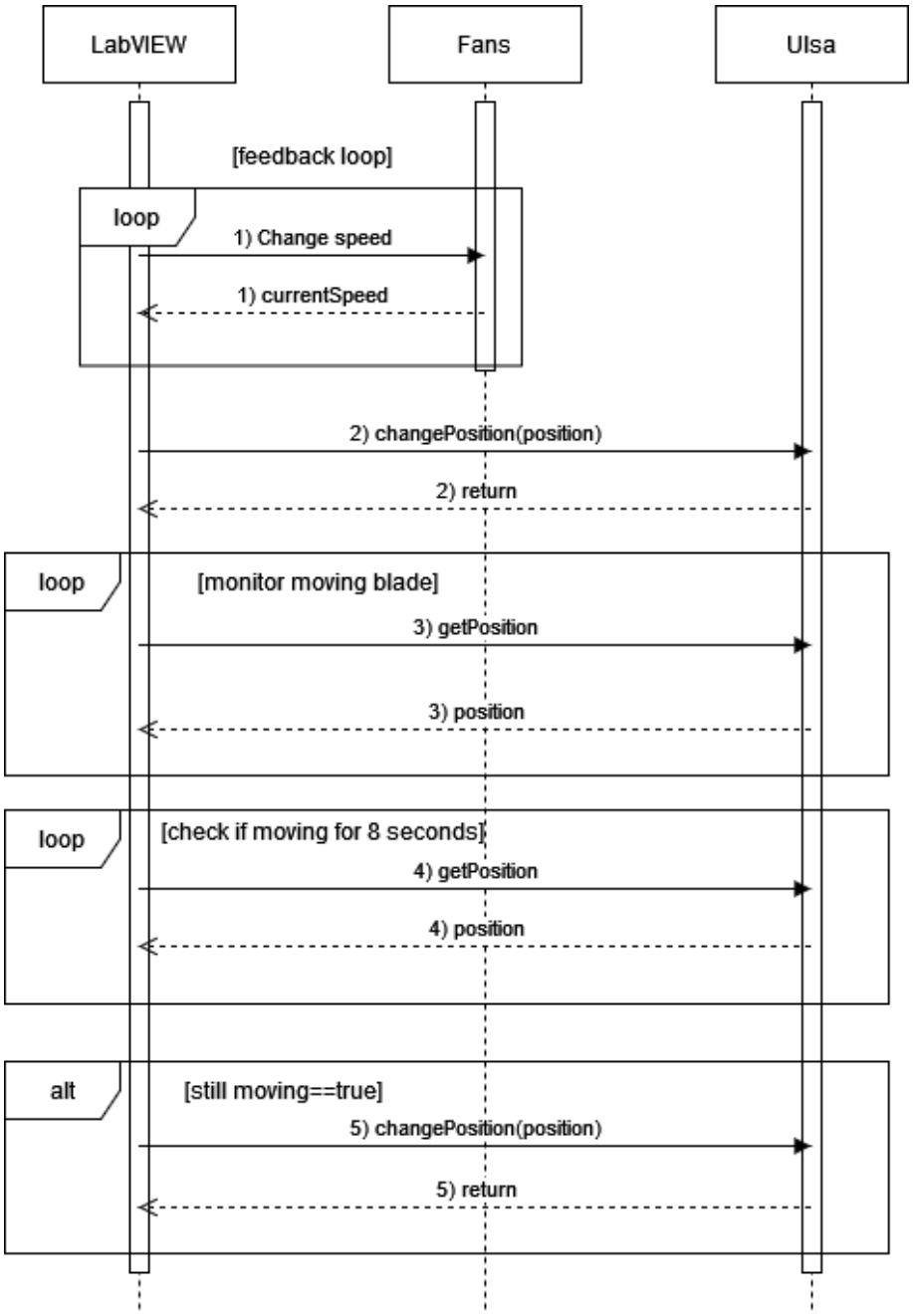

Figure 4.5. *Sequence diagram of Ulsa testing, Option B*

Figure 4.6 shows sequence diagram for option C. The figure has same actions as figure 4.4, but uses LabVIEW as a master module. Option C is the most complex one because a single command from LabVIEW to Ulsa goes through one more module than in option A.

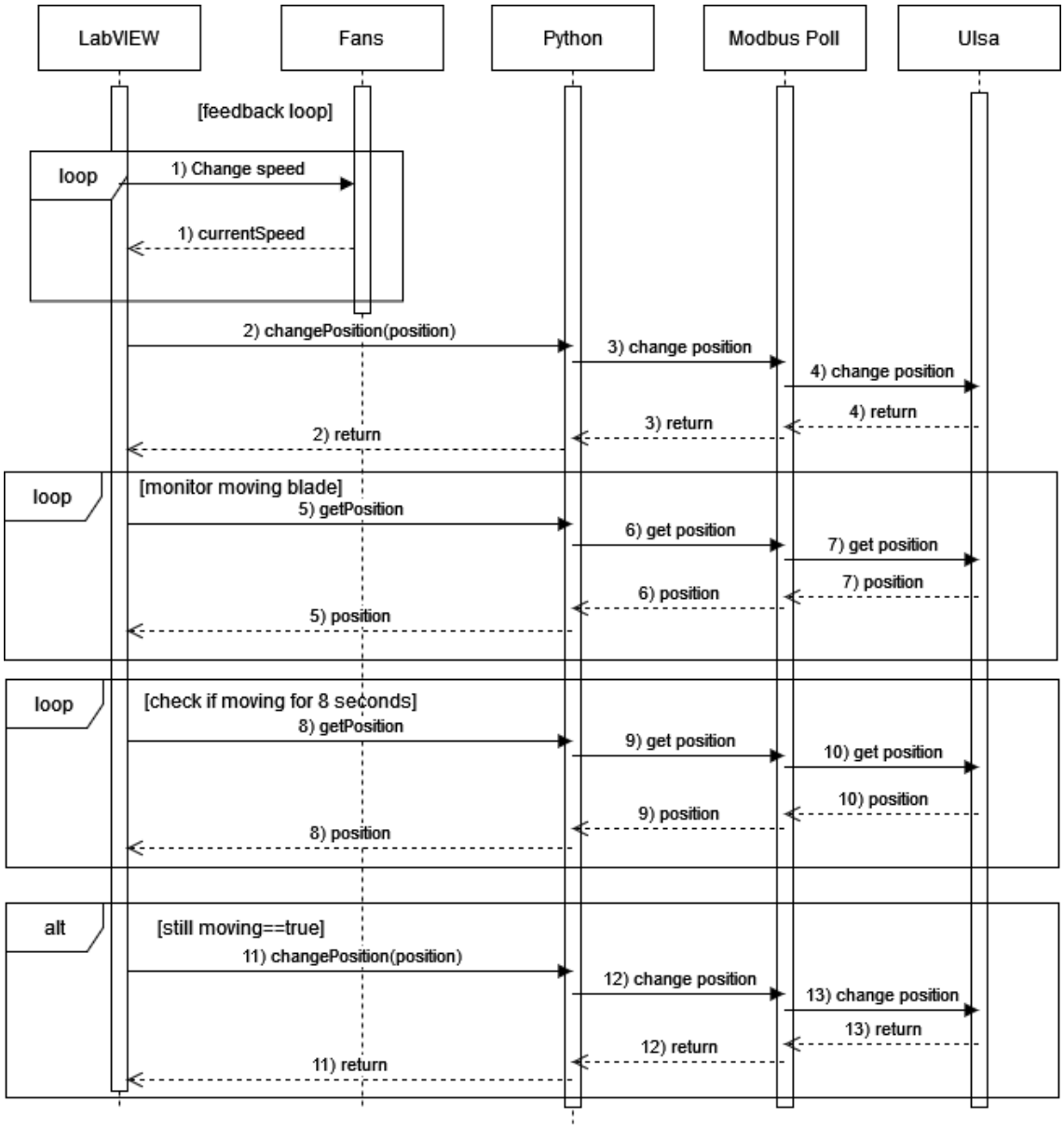

Figure 4.6. *Sequence diagram of Ulsa testing, Option C*

Figure 4.7 shows activity diagram for level 1 equipment testing. Yellow activities are related to fan control and monitoring and they are done manually. Follows same steps as done manually. Each fan speed uses same blade position, so the diagram loops with same fan speed as long as there are positions left to measure. After measuring every combination of speed and position the test will end. Logging of results is done every time after taking measurements. The results are gathered in a single Excel or CSV file, same data format as before. Measurement time can be adjusted.

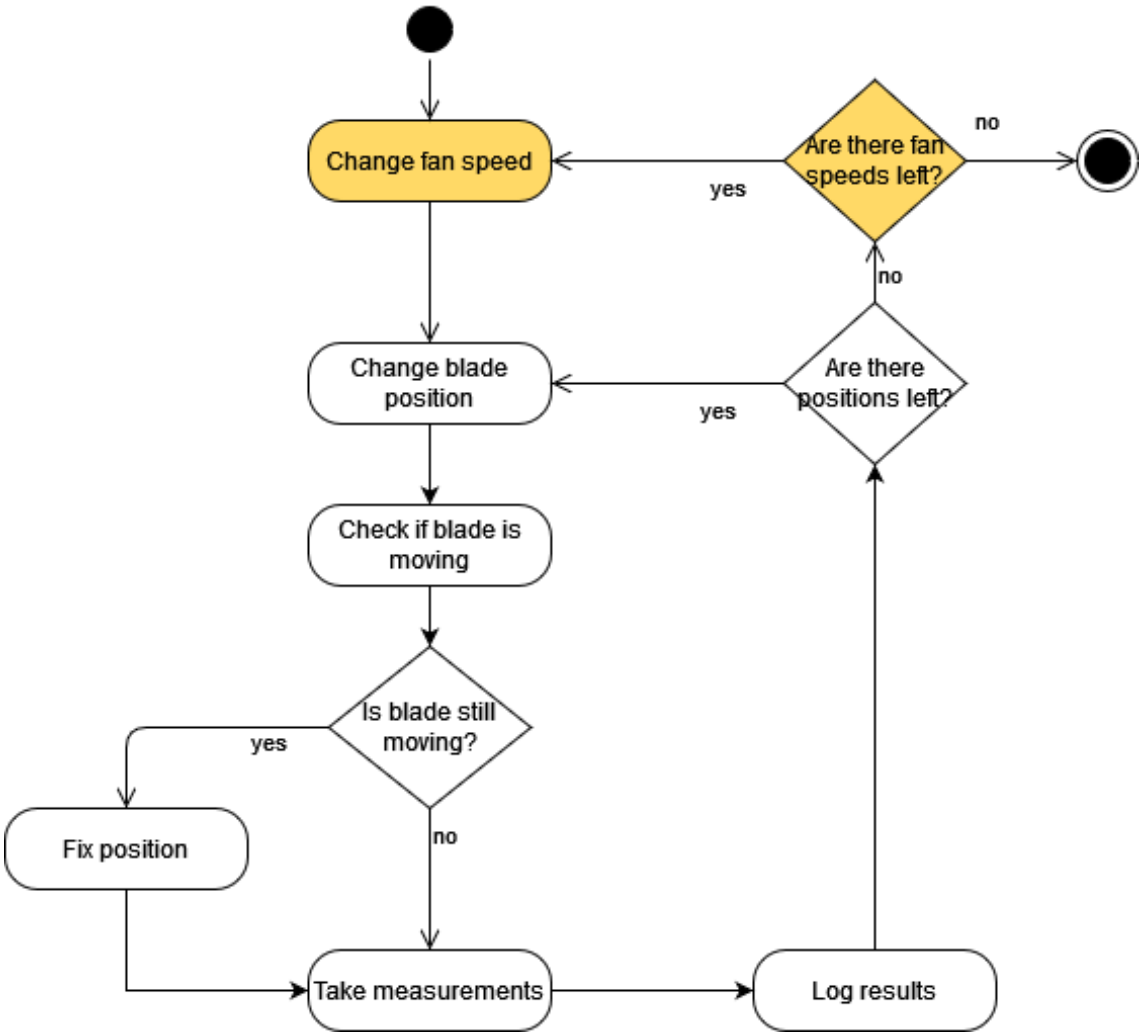

Figure 4.7. *Activity diagram of Ulsa testing. Yellow steps are done manually*

4.2.2 Level 2 equipment

Testing stays in black box testing because current test plan is sufficient and there's very little to be done to get access to Fico's inner workings. Tests stay in same level in V-model. Some tests can be automated if there would be more command options in Fico Pro program. The program needs update where user can input same commands as in GUI and if scripts could be run, more tests data could be generated. Time used for increased test data could be handled by scripts. Another way to automate some testing steps is to automate slave devices or create virtual versions of them. This equipment level's automation uses data-driven tests case presentation. Some test can be repeated with different inputs, otherwise the approach is more like capture and replay. All these tests will be in UI section of test automation pyramid, just like in old manual tests.

Figure 4.8 shows activity diagram for executing Fico's automated tests. The tests utilize scripts that use Tera Term macros. Each test is its own macro that master macro file executes. Using logic statements and output line checks [29] the state of Fico can be

determined. If state is unknown or not in a scope of test, error is logged and testing ends.

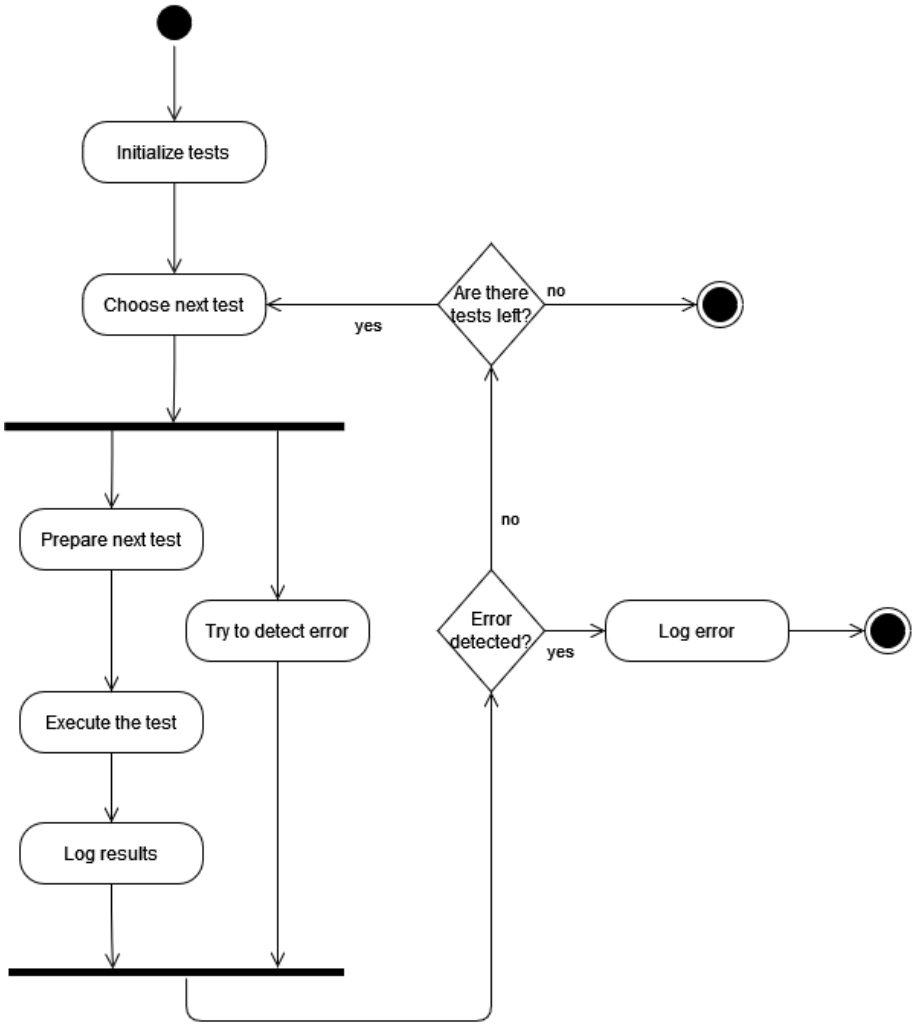

Figure 4.8. *Activity diagram for Fico testing*

Upgrading testing hardware to automate handling of slave devices and alarms. One PC would be connected to all devices, like alarm, slave devices and fire dampers. Not connected via Modbus because only one master device is allowed in a single bus. Hardware upgrades are out of limits of this thesis, cannot create software test solutions when manually done tests cover them already quite well. Slave devices could be virtualized. This would require to knowing all needed Modbus registers, so the virtual version of devices would be accurate. With virtual devices monitor and creating different room combinations would be easier.

Many test scenarios could use more test data. Now there's only one input in test data. More negative testing is welcomed. Adding scripts should do the work. Need of terminal/some way to input the test data. Using TCP version of Mobus makes it possible to use multiple masters in a single bus. Second master could monitor other devices.

Figure 4.9 shows two architecture designs for automated tests. It is updated version of

figure 3.11. In option A, a computer is connected to Fico's serial port via USB cable. Computer controls Fico with Tera Term. Tests are run from the computer and their results are gathered there. Option B differs from A in that the computer does not control Fico, but instead slave devices are virtualised inside the computer. Computer connects to Fico with Modbus.

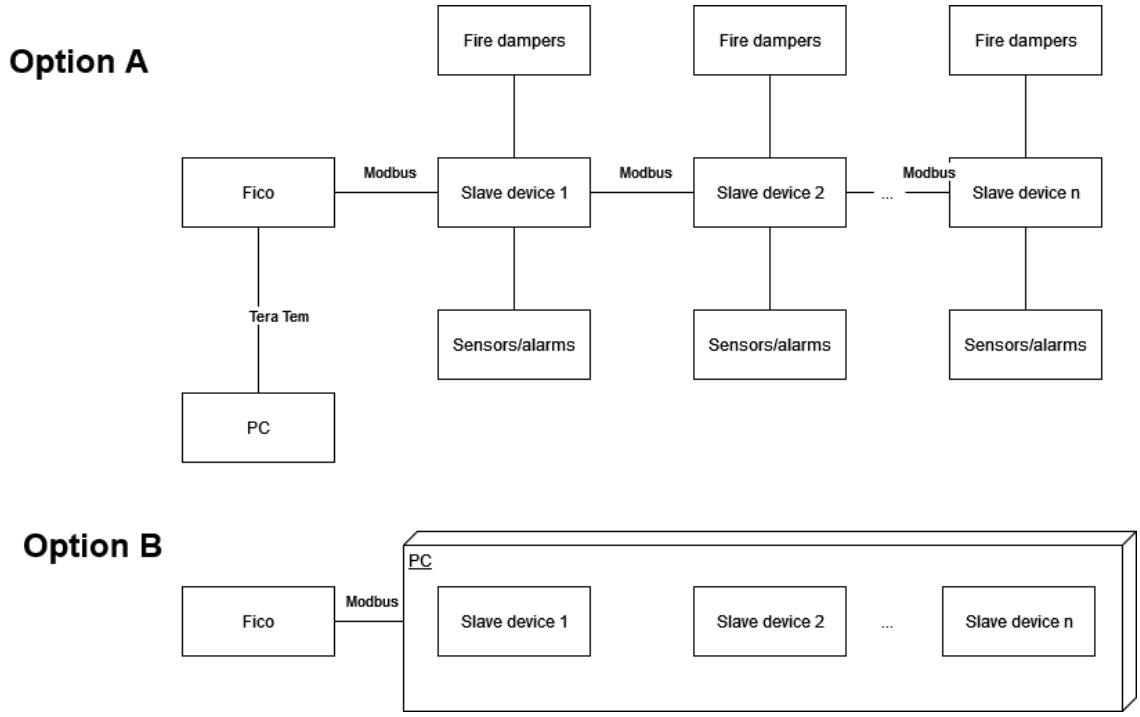

Figure 4.9. *Different options for automating Level 2 testing*

4.2.3 Level 3 equipment

Software versions 3.13 and 3.02 were analysed when finding tests. It's known that former has more bugs than the latter. The company's laboratory at Akaa only focuses on black box testing with this device. White box testing is done by software developer, but the company wants to tests more at their lab. In V-model new tests would be located in the bottom of the model, unit and integration tests. In automated tests pyramid tests are located in unit tests.

In source code there's a good function division, making the creation of unit tests easier. Functions' purpose and behaviour are documented in code. Creating integration tests is possible because the software is modular. It would be good to have access to software's documentation to get thorough insight of each software modules' requirements. To work correctly, some functions need to have a physical device connected to the software. Using the device in unit testing is an option, but simulating the device's behaviour would mean more automation and better control of tests. All this makes integration tests more complex to create. Virtualization would mean testing becomes hardware in loop testing. The tests

would use keyword-driven test case presentation. Same tests can be run with different inputs. Using same functions with different inputs is a major part in unit testing.

4.3 Test environment

In this section it is explained what tools and scripts tester needs for created tests and how to set them up. This section is divided into two parts. First part describes tools and scripts used in created tests. Tools are mostly same hardware used previously. New programs and programming libraries are required. Vital parts of scripts are explained to show tests' inner workings. Second part of this section describes test environment setup, what needs to be installed to PC and testing equipment are required.

4.3.1 Tools and scripts

Description of tools that are required to implement tests are given. Used vital algorithms and scripts are explained. Code snippets are collected from most important parts of codes and enlighten different syntax. Level 1 equipment requires licenses for LabVIEW and Modbus Poll. To connect Ulsa to PC a converter is needed. The converter is a UPort 1150I model and uses USB cable connected to one port RS-232/422/485 an adaptor with insulation. The Ulsa equipment uses RS-485 and it's same converter used in manual testing.

Python 3 is required and newest versions of Modbus Poll and LabVIEW. In Python, Pywin32 library needs to be installed. The test program can report the measurement to either SCV or Excel file, so licence for Excel is not necessary. Python PIP is also required, which is used to install Python libraries. Following option A from figure 4.2, figure 4.10 shows how Python can connect to Ulsa with Modbus Poll. Win32com is central to achieve this.

```python
import win32com.client as win32

App = win32.Dispatch('Mbpoll.Application')

App.Connection = 0            # Serial connection
App.SerialPort = 1            # Com port 1
App.BaudRate = 19200          # 19200 baud
App.Parity = 2                # Even parity
App.Mode = 0                  # RTU mode
App.ResponseTimeout = 1000    # Wait 1000ms until give up
App.DelayBetweenPolls = 20    # Ensure minimum 20 ms gap until next request
App.OpenConnection            # Open connection

# Create a Modbus display window called Win1
Win1 = win32.Dispatch("Mbpoll.Document")

# Read 10 holding registers from slave ID 1, address 0 (40001) every 1000ms
Win1.ReadHoldingRegisters(1, 0, 10, 1000)
# Show the Modbus window
Win1.ShowWindow()
# Show 10 rows
Win1.Rows(1)

# Disable refresh for speed
Win1.EnableRefresh = False
# Set the name of the registers
Win1.SetName(0, "Register 0")
# Set the value to write
Win1.EnableRefresh = True

Win1.ResizeAllColumns()
Win1.ResizeWindow()

_ = input("\nPress ENTER to quit ")
```

Figure 4.10. *Connect Python to Ulsa via Modbus*

In the figure Python is connected to Ulsa's first 10 holding registers. Before this connection parameters are defined, using serial port. The script creates a window in Modbus Poll to show these registers. This is just an example script, actual script used in testing is longer due to the need of handling more holding registers.

Figure 4.11 shows example code for connecting Python to LabVIEW [30]. In the example the code uses win32com library to connect Python to LabVIEW. First step of the script is to define file path to LabVIEW program. Second step is to set new values two LabVIEW's integer inputs. Row 18 runs the LabVIEW code. LabVIEW adds those two inputs together and save it to output value. After the LabVIEW finishes Python gets the output value in row 20. LabVIEW code must run to the end before Python can get return values. In current version of LabVIEW fan control the output values are updated in real time. This means the code must be modified if Python meant to be used.

```
1  import win32com.client
2  import os
3
4
5  Input1 = 100                                          # First Number to Add
6  Input2 = 200                                          # Second Number to Add
7  LabVIEW = win32com.client.Dispatch("Labview.Application")
8
9  filename = "testi_1.vi"
10 filepath = '{}{}{}'.format(os.getcwd(), os.sep, filename)   # Get full path
11 filepath = filepath.replace(os.sep, os.sep+os.sep)          # Replace \ with \\
12 print(filepath)
13
14 VI = LabVIEW.getvireference(filepath)                      # Path to LabVIEW VI
15 VI._FlagAsMethod("Call")                                   # Flag "Call" as Method
16 VI.setcontrolvalue('Input 1', str(Input1))                 # Set Input 1
17 VI.setcontrolvalue('Input 2', str(Input2))                 # Set Input 2
18 VI.Call()                                                 # Run the VI
19
20 result = VI.getcontrolvalue('Output 1')                   # Get return value
21 print(result)                                             # Print value to console
```

Figure 4.11. *Connect Python to LabVIEW*

For Level 2 equipment newest version Tera Term is installed. Figure 4.12 shows Tera Term macro that is used to connect and start Fico Pro program. Rows that starts with a symbol ';' are comments. Rows 1-3 and 29 handles data logging. Everything that prints to the terminal is saved to log file. Machine's password is saved in separate file and is retrieved in line 11. Lines 18-21 show that Linux commands work properly.

```
; Start logging
logopen 'test_log.log' 0 0
logwrite 'Log start'

; Connect
connect '/C=3'
setspeed 115200
sendln ''

; Login
getpassword 'password.dat' 'machine1' password
login='am335x-evm login' ; wait line that contains this text
wait login
sendln 'root'
wait 'Password'
sendln password

; Navigate terminal
rootText='root@am335x-evm'
wait rootText
sendln 'ls /'

; Open fico_pro
sendln '/home/app/fico_pro'
wait 'System started successfull'
send '1'
send '4'

logclose
```

Figure 4.12. *Tera Term macro for opening Fico program*

For level 3 equipment Atmel Studio is installed.

4.3.2 Test environment setup

First is level 1 equipment's test setup. A driver was installed named UPORT 1150 to the computer with Windows 10 operation system. With this the computer could receive and send Modbus data. Modbus Poll was used to make the computer act as a master in the bus. 227VM and ultrasound devices' slave ID is one. Ultrasound's sensor data can be collected from address 2071, with different rows to show the data. Ultra Ulsa was connected to computer via USB. Serial port converter converts the signal to readable data.

PyCharm is IDE for Python, which was used implementing tests. Other options are also possible, like VS Code. Need newest PIP in Python to install other libraries. Newest

version of Pywin32 is needed. After installing necessary Python libraries, the programs can be run from Window command window (cmd) or Git Bash. To run program, enter command 'py -u <path-to-root/file>'

Level 2 equipment setup computer is connected to Fico via USB cable. This computer runs scripts and logs test results. Physical setup configurations from old tests are used here, meaning nothing needs to be changed. Tera Term is installed. Computer can be connected to Fico using USB cable. A script shown in 4.12 can access Fico in Tera term. For tests that use virtual slaves, Modbus Poll Slave version must also be installed.

5. FINAL RESULTS AND ANALYSIS

All level 1 equipment's tests were executed in company's flow and system testing laboratory at Akaa with necessary testing equipment. No new hardware was acquired when old setup was good enough for new tests. Some new software was required. Automated tests were created only to level 1 equipment. Other levels are at planning or design level. Some software was new to the company. The tests were developed mostly elsewhere, and feedback gathered from test executions was used to improve the tests. Mostly same testing devices are used in new tests as in manual testing. Equipment was borrowed from the company for analysis. Because the testing is a constant process at the laboratory, it was easier to borrow extra equipment and analyse them. This equipment was also used in planning and developing new tests. Test results were collected at the laboratory and home and analysed at home.

Main specifications and detailed explanations of automated tests were given in previous chapter, so this chapter focuses on their results and analysis. First part of this chapter is about reviewing tests' compliance with original requirements. Each subgoal and equipment level are discussed separately. Second part of this chapter talks about future improvements for automated tests. Here suggestions are given how to make testing process better. The company is responsible to implement some of the improvements because they have the right skills for them. The last part focuses on personal thoughts about the work, its development, the company, the equipment, and tests. Learning outcomes are at a personal level.

5.1 Compliance with requirements

Final results are followed by their analysis. The focus is on answering this work's requirements mentioned at the beginning of this thesis. The goal is the requirement and subgoals add details to it. A summary of the goal is given after all subgoals are addressed.

Original goal of this thesis was to improve testing of equipment by automating some manually done tests. The goal was divided to four subgoals, the successes and problems of which are discussed here. First subgoal was to familiarize oneself with equipment and their tests. For levels 1 and 2, company's equipment was borrowed, so they could be analysed and used outside of company's working hours. Another reason for borrowing

was that it would have been harder to try out the equipment when tests were being executed. Technical information was gathered from company's website, like Modbus communication, manuals, and installations. The subgoal included interviewing testers and monitoring their test executions. Testers explained steps required in tests and gave info about the equipment. More technical information about Fico's slave devices could have been collected for later analysis. With these two equipment levels, the subgoal was a success. How well the equipment was analysed comes apparent in later subgoals. Because unit tests were points of interest in level 3 equipment, the focus was on source code. The ultrasound device is part of original tests, so analysing it might have brought more knowledge about the tests. No original system tests were reviewed. Considering these facts, the subgoal with level 3 equipment was only partly success.

Second subgoal was to analyse old tests. The idea was to search possibilities and limitations for test automation. Testers were interviewed once more, but this time it was about what to automate. Because the number of tests to automate, they were prioritized and criteria for this was created. These focused on how much tester's time could be saved and how much automation would affect test maintainability. In level 1 Ulsa's tests were selected, leaving tests for 227VM behind. Ulsa's testing is repetitive, so automating it would save more time than 227VM's tests. Basic idea behind discovered steps in manual testing was unchanged and each step was analysed.

Level 2 analysis were different from level 1. Testing methods were different and so was their analysis. One tester showed how to get access to Fico's program and monitor it. It opened possibilities for automated input and output handling. Slave devices were hardly analysed because not much information was collected about them in the previous step. Only how to trigger alarms and read devices' status were noted. Prioritization criteria were not used in this level because automating Fico's interface would mean most of tests or part of their steps could be automated. For level 3 no analysis was done for old tests because there were no unit tests. Overall, the second subgoal was successful because it created enough information for the next step.

Third subgoal was to design automated tests. Original idea was to mainly use resources used in old tests. Some design solutions require new software, but in these cases maintainability and level of automation rises. Level 1 test design gave three options. They use same hardware as before but with different software solutions. One option is differing from other two, with Python as a master module in test program. Others have LabVIEW as a master. Also, options were given on how to implement fan's feedback control. Previously the control had to be manually adjusted. Level 2 test design gave two ways to automate the tests, one is Fico program's software update and other is using virtual slaves instead of physical ones. No white box testing was designed because Fico's inner working were not analysed. Level 3 test design didn't give many results. General info was given on how to implement unit tests, but possible options were not researched.

Overall, the third subgoal was successful with levels 1 and 2. Different ways to automate and what old and new resources were needed for them were presented. In these solutions Modbus was prominently displayed in automation. Level 3 was left behind, with no clear design plans created. White box testing of level 2 was left untouched because the laboratory only practised black box tests.

The last subgoal was to implement automated tests. In the beginning it was clear that not everything was possible to be implemented. For this reason, the subgoal was considered optional. Level 1 was the only level that included implementation. One of the three design options was chosen according to author's skills and available resources. The chosen design was option A from figure 4.2. Steps that included fan control weren't automated. Either way, the current level of automation saved tester's time an estimated a few minutes. Multiplying this by 400 saves many hours. Overall, the last subgoal was achieved only at level 1.

The goal as a whole was partially successful. Tests were improved by automation. In design phase it was shown that automating test execution and test results handling saves tester's time and opens possibilities to improve the testing process. Every subgoal was revisited when new information was needed or improvements for designs were found. In level 1 all subgoals were reached. Some automation was implemented, and few additional automation options were designed. In level 2 automation options were found for two major topics. Some design was done and would need more information to proceed further. Level 2 would need updates mentioned earlier to work. Plans were made for levels 2 and 3 to reach the implementation subgoal. Some of these updates belong outside of this thesis. Level 3 was a failure. Very little analysis and design was done to the ultrasound device.

Overall, the goal was reached with equipment levels 1 and 2, with level 1 also including implementation phase. Also, in level 1 the test design was more detailed than in others. Only level 3 was nearly untouched throughout the work, almost as it could have been removed and had no effect to the end results.

5.2 Future improvements

This part of the chapter handles two kinds of improvements. First type focuses on improvements to designed automated tests and second one on improvements that the company can implement to their equipment and testing. To get full benefits of automation, improvements done by the company are more important. Some improvements can be done without company's input, but the most important updates require the company. First improvements that effect all equipment levels' testing are discussed, then each level separately. Then by each level.

There are several improvements that affect all equipment levels. First two are about

documentation, and later ones use of time in thesis. When the LabVIEW code for the fan control was reviewed, there was no general documentation. Creating a manual for all LabVIEW code would help other coders to understand the code better. Simply as explaining interfaces and primary behaviours and purpose would be a start. Code's structure, activity and behaviour can be visualised with UML diagrams. Another improvement for documentation is test plans. Testers needed to be interviewed because there was none or very little about test setups. The testers knew what they were doing but a lot they told were not found in documentation.

Exploring more software and hardware options for automation could have brought more suggestions. Some suggested licenses for LabVIEW were quite expensive, so searching alternatives could reveal more economical viable add-ons. All the software mentioned in automating levels 2 and 3 are not necessary the only options for testing. They were just easiest to acquire and bring into use. Interviews were unstructured and poorly planned. An interview plan with equipment analysis and list of questions would have been more efficient than used 'ask when questions arise during manual testing.

All version control for code was done in a way that could be considered anti-coding. One option would be GitLab, which offers repositories for code projects. It shows code changes and allow branches. Branching allows multiple people to program different features at the same time.

Improvements for level 1 includes implementing other automation options, automating fan control and designing tests for 227VM actuator. Current version of automated tests, option A from figure 4.2, requires tester to adjust fan speeds manually. Python executes LabVIEW program updates fans' info in GUI in realtime. When Python executes LabVIEW program, end results can be gathered after LabVIEW program ends. Automating fan control and make it part of current tests would increase automation. It would also save tester's time when the whole process found in figure 4.7 could be done without tester's input.

Two latter design options from figure 4.2 require LabVIEW as a main component. Studying LabVIEW's would help implementing these options. Creating tests fully with LabVIEW because company uses it a lot, so maintaining tests wouldn't be hard. In this case option B would be the best because it doesn't use Python.

227VM tests were untouched because of time limit of this thesis. Those tests could be improved: Analyse 227VM tests and design automated tests. Automating these tests would modify original plan because the tests would require Modbus to make them work. In original plan 227VM's state is read from its front interface.

In level 2 equipment testing there are three main issues to improve. First one is about Fico program. It could be controlled from computer with Tera Term, but the program didn't

have much of an interface. It only printed some information about slave devices' status. Updating the interface to allow better monitoring and control would allow automated tests. Second issue to improve is about slave devices. Either control and monitor them from computer or create virtual version of them. This would mean virtualised hardware-in-the-loop testing. This issue requires slave devices' Modbus documentation. A project this size would be a thesis itself. If all issues are completed, tests can be run multiple times with different order. This would take too long if done manually.

Third issue is about combining two previous improvements. Because Tera Term cannot communicate with Modbus Poll Slave program or Python, implementing both automation options from figure 4.9 is not possible. Python can control the slave program and devices' serial ports. So replacing Tera Term with Python would mean both automation options can be used in the same testing setup.

With those three issues Fico's testing could be fully automated. Solving just one issue doesn't remove tester's role from test execution. Only updating Fico's interfaces means the testers must control slave devices and gather their data manually. Only implementing slave devices would mean the tester must use GUI for Fico's inputs and outputs. Either way, each option adds level of automation.

Only black box testing was done to Fico, when originally white box testing should have been included. To design these tests documentation to Fico program's inner working and source code would be needed.

There are also other minor improvements. Tera Term used in Ficos's test design is sufficient when it comes to scripting, but maintainability can become problem. New functionalities and changing requirements mean the code must be kept updated. A possible solution to this would be designing a modular script from which tests can be detached, moved, and added. This approach is referred as keyword test case presentation. Tera Term can use wait times if program's process must achieve its goal in certain time. This means a script can be interrupted if the process takes too long. Without wait times Tera Term can wait indefinitely for process' end. Wait times must be tested and adjusted for each test. Expand information gathering, like interviewing testers, outside of Akaa laboratories. Because a product's development is divided to multiple places, gathering information from these would be helpful.

Improvements for level 3 includes studying interfaces and physical device. The most important improvement is to contact and interview software developer of the device. This is one way to get better handle on documentation and system requirements. Familiarise oneself more with C language and Atmel Studio. Gather information about unit testing in that environment and search possible alternatives for Atmel if testing cannot be done with it. Focusing on hardware-in-the-loop would be a good solution for system testing. In the thesis there's was no studying the physical equipment like in other levels. This could

open more options for testing.

5.3 Personal thoughts

I wanted to see the world outside of university. In university studies each project was well supervised, the outcome was predictable, the project was divided into parts with clear goals, and software was a priority in the projects. Working on this thesis for a company meant all those issues changed. The work was supervised but the responsibility was on me alone. Outcome was not fully known because all technology and testing was unknown to me. In course studies a project had clear goals with each deadline. In this work goals with their deadlines were constantly updated when more information was gathered about the equipment. In tests, the hardware was as important as software and it was completely new experience.

With three devices in focus, thesis' topic was quite broad. Every equipment's automation could be a thesis itself. This way equipment could have been studied more closely. Studying Fico's and ultrasound device's software would require more time and resources. Results of this thesis are more like first versions test designs and prototypes of test implementations.

Approach to test automation was interesting. Analysing old manually done tests and then automate them. Normally the automation relies on documented requirements, but in this case the requirements were gathered from manual tests. It was first time I was introduced to hardware testing. It's different from software testing with its requirements and testing methods. Once again UML diagrams were helpful when it came to describe structure and behaviour of tests. It makes easier to reader to understand concept of the systems. UML could be used more in the future.

Realized it's more important to recognize what parts of test can be automate than trying to automate the whole testing process. It's good to start from small parts and then expand the automation. Don't invent the wheel twice, use already existing tools, libraries, and code to implement automation. Recognize what kind of tools the company uses and supports. This may have impact on the tools can be used. Automation is not silver bullet, not all tests should or could be automated. It's important to listen and ask testers about automation options. Interview programmers to get information about used resources. There's quite difference between written documentation and verbal communication. Personally, I have never conducted interviews on this scale before.

6. CONCLUSION

The purpose of this thesis was to automate tests of ventilation equipment. They are products of a company called FläktGroup. Automated testing means that a software can run tests and collect test reports instead of testers. Automation's purpose is to save time and ensure testing is performed as planned, without human errors. The thesis focused on three devices and their tests. The equipment is divided into three levels, each containing one device. These levels were

- Level 1: Ultra Ulsa, a VAV damper
- Level 2: Fico Pro, a fire damper control and monitoring system
- Level 3: Ultrasound air flow measurement device

A goal was created to help the thesis' purpose. The goal was to automate these equipment's tests. It was divided to several subgoals, and thesis' structure was built according to them. Before analysing tests it was important to understand what is the purpose of testing, how to approach it, and how testing is used in project development.

Next was an introduction to the devices and their tests. The company offered equipment, devices that are part of testing, and test laboratory. After getting acquainted with the tests, they were analysed to gain a better understanding of the possibilities and limitations for automating them. The testing process for each equipment level was different.

Next step was to design new tests. A few design options were created, each solution using different set of software and hardware. These gave various possibilities to implementation, for each option requires different amount of money and skill. A large number of tests and limited amount of time meant that only some tests could be automated. Criteria were created for prioritize the tests. Primary criteria was how much an automated test can save time. This meant speeding up the test execution and saving testers from repetitive tasks. Secondary criteria was maintainability of new tests. Interviewing testers helped in this problem.

While designing automated tests, suggestions and future improvements were gathered for both the company and this thesis. These were collected during design and result analysis phases. Improvement for company includes new software and hardware upgrades, documentation, and upgrades to interfaces so it's easier to run scripts on the software.

End results of automated tests were promising. Most of the designs use company's already existing software and hardware. Some new software was suggested in design options. Some level 1 equipment's tests were implemented. New tests save couple of minutes of testers time, but considering these tests are repeated hundreds of times, total time saved is measured in hours. Level 2 test designs focused on updating interface so running scripts was possible and virtualisation of hardware for easier and faster input and output handling. Level 3 was nearly untouched. Only some basic info and planning was gathered. The main goal was achieved with levels 1 and 2.

REFERENCES

[1] Homès, B. *Fundamentals of software testing*. eng. Iste. Hoboken, New Jersey: ISTE/Wiley, 2012. ISBN: 1-118-60227-7.

[2] Spillner, A. and Linz, T. *Software Testing Foundations, 5th Edition: A Study Guide for the Certified Tester Exam*. eng. Rocky Nook, 2021. ISBN: 9781681988535.

[3] Borba, P., Cavalcanti, A., Sampaio, A. and Woodcook, J. *Testing Techniques in Software Engineering: Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures*. eng. Vol. 6153. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2010. ISBN: 3642143342.

[4] Buede, D. M. *The engineering design of systems : models and methods*. eng. 3rd ed. Wiley Series in Systems Engineering and Management. Hoboken, New Jersey: Wiley, 2016. ISBN: 1-119-02806-X.

[5] Copeland, L. *A Practitioner's Guide to Software Test Design.* eng. Artech House computing library A practitioner's guide to software test design. Norwood: Artech House, 2003. ISBN: 1-58053-732-4.

[6] Mosley, D. J. *Just enough software test automation*. eng. 1st edition. Upper Saddle River, NJ: Yourdon Press, 2002. ISBN: 0-13-008468-9.

[7] Subramanian, C. *Software engineering*. eng. 1st edition. Chennai: Pearson India Education Services, 2016. ISBN: 9789332558298.

[8] Games, T. *Pairwise Online Tool*. `https://pairwise.teremokgames.com/`. Accessed: 2021-10-27.

[9] Pajankar, A. *Python Unit Test Automation Practical Techniques for Python Developers and Testers*. eng. 1st ed. 2017. Berkeley, CA: Apress, 2017. ISBN: 1-4842-2676-3.

[10] Baumgartner, M., Klonk, M., Mastnak, C., Pichler, H., Seidl, R. and Tanczos, S. *Agile Testing: The Agile Way to Quality*. eng. Cham: Springer International Publishing AG, 2021. ISBN: 9783030732080.

[11] Kleijn, C. *Introduction to Hardware-in-the-Loop Simulation*. `www.hil-simulation.com/images/stories/Documents/Introduction%20to%20Hardware-in-the-Loop%20Simulation.pdf`. Accessed: 2022-01-19.

[12] *Modbus Organization. Modbus FAQ: About The Modbus Organization*. `https://modbus.org/faq.php`. Accessed: 2021-10-15.

[13] Seneviratne, P. Modbus. eng. *Building Arduino PLCs*. Berkeley, CA: Apress, 2017, pp. 127–138. ISBN: 1484226313.

[14] *Modbus Organization. Modbus Serial Line Protocol and Implementation Guide V1.02*. https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf. Accessed: 2021-10-15.

[15] *Real Time Automation. An Introduction to MODBUS RTU*. http://www.rtautomation.com/technologies/modbus-rtu/. Accessed: 2021-11-16.

[16] Belliardi, R. and Neubert, R. Modbus Protocol. eng. *Industrial Communication Technology Handbook*. 2nd ed. CRC Press, 2015, pp. 10-1-10–34. ISBN: 148220732X.

[17] Black, R. *Managing the testing process: practical tools and techniques for managing software and hardware testing*. eng. Wiley-Blackwell, 2011. ISBN: 9780470404157.

[18] Wheat, D. *Building Your Own Electronics Lab A Guide to Setting Up Your Own Gadget Workshop*. eng. 1st ed. 2012. Technology in action. Berkeley, CA: Apress, 2012. ISBN: 1-4302-4387-2.

[19] FläktGroup. *227VM-MB Optivent Controls Technical catalogue*. https://www.flaktgroup.com/api/v1/Documents/2073e64c-1010-4aef-af4b-941c6efb1b2f?search=0. Accessed: 2021-10-05.

[20] FläktGroup. *Optivent Ultra VAV damper Technical Data*. https://www.flaktgroup.com/api/v1/Documents/e2cf70d2-bba5-416f-bc98-f26fcce9d32c?analytics=0. Accessed: 2021-12-09.

[21] FläktGroup. *227VM-MB Compact Controller For Modbus*. https://www.flaktgroup.com/uk/products/air-management-atds/controllers/variable-air-volume-controllers/227vm-mb-compact-controller-for-modbus/. Accessed: 2021-10-05.

[22] FläktGroup. *Optivent Ultra Control unit 227VMZ-MB*. https://www.flaktgroup.com/api/v1/Documents/9cbd8c61-add2-4d73-954d-8a38167fd724?analytics=0. Accessed: 2021-12-14.

[23] *National Instruments Corp. What Is LabVIEW?* https://www.ni.com/fi-fi/shop/labview.html. Accessed: 2022-01-17.

[24] FläktGroup. *ISYteq FICO Flyer*. https://www.flaktgroup.com/api/v1/Documents/a7b7ed10-0e0e-42b7-b81c-a1f95e8d19cc?analytics=0. Accessed: 2021-12-13.

[25] FläktGroup. *Ohjausjärjestelmä ISYteq FICO-PRO*. https://www.flaktgroup.com/fi/products/ilman-hallinta-ja-huonelaitteet/palopellit/palopeltien-ohjausjarjestelmat/ohjausjarjestelma-isyteq-fico-pro/. Accessed: 2021-11-18.

[26] FläktGroup. *Ultrasound Flyer*. https://www.flaktgroup.com/api/v1/Documents/31985cfc-fc8d-40a8-9bfb-889a7353fd46?analytics=0. Accessed: 2022-01-18.

[27] Teranishi, T. *Tera Term Open Source Project*. https://ttssh2.osdn.jp/index.html.en. Accessed: 2021-12-02.

[28] Halvorsen, H.-P. *Modbus With Practical LabVIEW Excamples*. https://www.halvorsen.blog/documents/technology/modbus/modbus.php. Accessed: 2022-01-17.

[29] *Softpanorama Society. TeraTerm Macros*. http://www.softpanorama.org/Utilities/Teraterm/teraterm_macros.shtml. Accessed: 2022-01-16.

[30] *National Instruments Corp. Controlling a VI using Python using LabVIEW*. https://forums.ni.com/t5/Example-Code/Controlling-a-VI-using-Python-using-LabVIEW/ta-p/3536468?profile.language=en. Accessed: 2022-01-16.