Expertise
and insight
for the future

Tiia Rautavesi

Test automation as a part of agile software development

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and communications technology

Bachelor’s Thesis

10 May 2019

Metropolia
University of Applied Sciences

Author Title Number of Pages Date	Tiia Rautavesi Test automation as a part of agile software development 36 pages + 1 appendix 10 May 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Media technology
Instructors	Jussi Latvaniemi, Project Manager Aarne Klemetti, Researching Lecturer

This research project was a case study about implementing test automation as a part of agile software development to the client's software testing processes. The main research problem was to find the necessary improvements for the testing process to make it more efficient and optimized by test automation in sustainable way.

This thesis inspects the differences and benefits of test management tools in Waterfall and Agile software development models and used the collected information for creating the best possible test process for the client.

This thesis aimed to find the best test automation framework and test environment for the client to use for test automation. The client needs a test automation framework that is easy to use and can be used for modular testing. It also has to be free and based on open-source.

The structure of the new test process is mainly based on Agile model and its principles about involving the customer and end user. The test process is updated by adding documentation about user stories and technical requirements. This documentation will work as the basis for test automation. Robot Framework was selected as the test automation framework, because it supports modularity and is easy to use.

The new testing process that we recommend is built around user's needs. Test automation will create a more reliable, faster and repeatable test process. Robot Framework will ease out the process of collecting test data as it provides tools for creating detailed test reports.

Keywords	Test Automation, Robot Framework, Agile, Software development, Software Testing, Test Management

Tekijä Otsikko Sivumäärä Aika	Tiia Rautavesi Testiautomaatio osana ketterää ohjelmistokehitystä 36 sivua + 1 liite 10.5.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	mediatekniikka
Ohjaajat	projektipäällikkö Jussi Latvaniemi tutkijaopettaja Aarne Klemetti

Insinöörityössä tutkittiin testiautomaation sisällytystä ja ylläpitämistä osana asiakkaan ketterää ohjelmistokehitystä. Lähtötilanteessa asiakkaalla ei ollut vielä testiautomaatiota sisällytettynä prosesseissaan ja ohjelmistotestaus oli hajanaista. Tutkimusongelma oli asiakkaan ohjelmistotestauksen jäsentäminen ja parantaminen ja testiautomaation sisällyttäminen yrityksen prosesseihin ketterästi ja kestävästi.

Työssä tutkittiin ja vertailtiin vesiputousmallin ja ketterän kehityksen (agile) mallin tarjoamia periaatteita ohjelmistotestauksen näkökulmasta. Näihin menetelmiin ja niiden esittämiin testausmalleihin pohjautuen asiakkaalle luotiin uusi testiprosessimalli, johon sisällytettiin tarvittavat työvaiheet ja perustiedot menestyksekkään testiprosessin täytäntöön panemiseksi.

Työssä perehdyttiin testiautomaatioon myös yleisemmin, mutta pääosin tapauskohtaisesti, asiakkaan näkökulmasta. Asiakkaan tarpeisiin etsittiin parasta mahdollista testiautomaatiokehystä. Tärkeimmät vaatimukset valittavalle ojelmistokehykselle olivat modulaarisen testauksen tukeminen ja helppokäyttöisyys. Sen tuli myös olla ilmainen ja avoimeen lähdekoodiin perustuva.

Tulokset testiprosessin uudistamisen kannalta pohjautuivat pitkälti ketterän kehityksen malliin ja sen periaatteisiin asiakkaan ja loppukäyttäjän huomioimisen näkökulmasta. Asiakkaan prosessien aiemman tilanteen perusteella niihin oli sisällytettävä aiempaa enemmän kommunikaatiota ja dokumentaatiota asiakkaan tarpeista. Testidokumentaatiota painotettiin lähinnä käyttäjätarinoiden luomiseen ja teknisten vaatimusten luomiseen niiden pohjalta. Tämä projektin tai sprintin alussa luotu dokumentaatio taas toimii alustana tuotekehitykselle ja testiautomaatiolle. Testiautomaatio luodaan RobotLab-ympäristössä Robot Framework -ohjelmistokehystä käyttäen. Robot Framework valittiin testiautomaatiokehykseksi, koska se tukee modulaarisuutta todella hyvin ja se on helppo ottaa käyttöön varsinkin RobotLabin kanssa. RobotLab mahdollistaa testien luomisen ja jakamisen selkeässä muodossa, ja testit voi paketoida ja jakaa linkin avulla muille tiimin jäsenille, jotka voivat ajaa testejä ilman RobotLabin lataamista.

Insinöörityönä tehty uusi testiprosessi ottaa asiakkaan tarpeet paremmin huomioon kuin aiempi. Testaaminen on myös luotettavampaa, nopeampaa ja helpommin toistettavaa testiautomaation sisällytyksen ansiosta. Testidataa on huomattavasti entistä helpompaa kerätä, sillä Robot Framework tuottaa yksityiskohtaisia raportteja testeistä.

Avainsanat	testiautomaatio, Robot Framework, ketterä kehitys, agile, ohjelmistokehitys, ohjelmistotestaus

Contents

List of Abbreviations

Static testing	Testing a static software draft that is not produced yet
Dynamic testing	Testing a software product that has been developed
White box testing	Testing a software product without without insight to the code
Black box testing	Testing a software product with background knowledge of the code or access to inspect the code
Test case	Entity of information about test inputs, conditions and expected results of the test
Test suite	Combination of test cases that are run one after another, usually in the same order each time
DSL	Domain Specific Language
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JavaScript	Programming language that is commonly used in web environments
GitHub	Online platform for hosting software and for version control
Repository	A data structure for storing software files or directories and their metadata

Jupyter Notebook	Web application for creating and sharing live code. In this thesis it's the basis for JupyterLab which is the basis for RobotLab.
RobotLab	JupyterLab with Robot kernell installed. Used as a graphical user interface for Robot Framework.
SaaS	Software as a Service

1 Introduction

This research project is a case study about creating efficient and maintainable test processes for the most common test cases of the client. It aims to optimize the existing and the new test processes by implementing test automation in the most usable and agile way possible. The project proposes clear instructions and useful technical tools that help to keep the testing structured. With the help of these tools the members of the company can use and maintain the test processes independently in the future.

The research and technical execution was ordered by a small Finnish IT-company. The client company focuses on creating tailored e-learning solutions for customers in multiple business fields. Their company has mainly grown around a software application, which offers the customers a chance to create and edit their own e-learning modules. Currently there are over 100 active modules that have dependencies to this software, which means over 100 modules that must be tested each time a major change is implemented to the main software.

The client selected this topic due to the unstructured nature of the current software testing processes. Currently most tests are unstructured and minimally instructed and documented. They are manual tests that are performed after building the software, which could be described as the most expensive way to test a software product. It is expensive, because the product does not get verified and validated continuously, but instead only after the development phase. Therefore, if the test results show that the developed product can not be verified to match the requirements, the time and price of development can be doubled. Moreover, the tests are often performed by testers who are not familiar with the requirements and therefore the testers can produce more bug reports by mistaking features to bugs and elongate the development process that way.

This research project aspires to find the tools that the client needs for increasing the tests' reliability and effectiveness, increasing quality of service delivery and decreasing the testing costs. The study also inspects ways to increase the test documentation and, by reflecting the found options to the client specific situation, tries to find the optional test documentation standard. Besides confirming that the final product matches the requirements, successful testing includes generating test data that can lead to improvements in both usability and technical execution of the end product. The test data is extremely important when a bug is found, to see when was the last time and what the software environment was like when the feature worked and the bug did not exist.

As a method to find a solution for the previously stated problems and needs, we decided to research implementation of test automation in the client company. The popularity of test automation is based on its efficiency, repeatability, speed and affordability. Test automation offers convenient features for documenting requirements, test plans, test results and more specific test data. In the present day test automation relies quite heavily on writing test scripts manually and it has to be maintained and updated manually as the software keeps evolving. In this research project we tackle this issue with modular approach to automated testing, that ensures that the tests are reusable and effortless to execute and update when needed.

The question this research project is trying to answer is the following:

- What kind of test process is the best for the client, how can it be optimized by test automation and how will it be maintained in agile software development environment?

This question combines the three main research areas. First is finding the right test theory and process for the client’s specific needs. Methods of the researching this are are theoretical, such as research of the client’s specific problems and finding solutions by comparing different testing models and test automation tools. Second area –test automation is researched for concretical, technical point of view and tailored to test the current software product. This are demands knowledge of the product and research methods such as interviews of the development team and studying the software structure, in both code and architectural sides, are very important. Third area is the implementation and maintenance of the test process and test automation tools.

The results of this change in test processes can be analysed by measuring uptime of the software, amount of bug reports from the customers and money saved on testing costs by reduction of time it takes to test a certain piece of software. Customer satisfaction can be measured by literal feedback that can be collected with surveys or conversations.

2 Structured testing

First method for finding the best test process was inspecting different structured testing models. We inspected multiple models for structured testing and test management in software development environments. These models provided essential information and tools for creating regulated test environment for the testers and other members of the development team. This chapter takes a look at the most common testing models and reflects the client's processes on them. The goal is to find a framework that could provide a base structure for stable and efficient testing processes that benefits the client's needs.

In this project it was necessary to pay attention to the communication between different roles within the production team. Even though Agile Manifesto (1) talks against documentation, it can be very important depending on the test case. Verbal sharing of requirements and test results can be enough for the most minor features, but in large projects the lack of documentation can lead to problems. Testers have to return data to the production, so that the product can be improved, and possible imperfections can be fixed. The documentation of the tests and test results does not need to be strictly formatted or excessive, but there has to be communication about the problems or improvement ideas that come up during the testing. It is also important to keep track on when a bug did or did not exsist. Another point of view includes adding documentation in the inception of the project by creating requirements that can be referred while development and testing.

The client has multiple situations in their development processes where testing is well integrated already, and this projects seeks to create further structure to these testing scenarios and improve the test process by adding test automation as an important tool, that can reduce work hours spent on testing. At very least it helps these work hours to be used for increasing the quality by going into details more than previously.

There are several ways to test a software product. When evaluating whether or not the software meets the requirements specifications, two measures used are verification and validation. According to SQA’s material about verification and vlidation, validation answers to the question: ”Are we building the right product?” and verification to the question: ”Are we building the product right?” (2). One of the main tools for validation is static testing, which means testing a product without running the program. It is a way for finding what the user actually needs and figuring out how the product could be improved to match these needs more than before. Static testing is often used for testing the usability of the product. The *Guru99*’s article *Static and dynamic testing* explains that dynamic testing involves running the program and it can for example be used for acceptance testing. In practice acceptance testing can involve inspecting the entire system on weather it meets the requirements or not (3). Dynamic testing can be divided in black box testing and white box testing. Described by the *Software Testing Fundamentals* black box testing means that the tester is not familiar with the system structure, such as code (4). Black box testing imitates the user’s point of view and does not take the quality of the code in consideration. Software Testing Fundamentals describes that white box testing in the other hand includes inspecting the code too and observes the software from the developer’s point of view. White box testing can be executed before the graphical user interface is ready, as it focuses mainly on structural and functional aspects of the software (5).

2.1 Differences and benefits of Waterfall model and Agile model in software testing

In the present day software development is mainly based either on traditional Waterfall model or more modern Agile model. Both of these serve well in certain projects, but as the technical development keeps only getting faster it is easy to see the benefits of Agile development methods. This chapter goes through the differences and benefits of these models and how they approach and include testing. These results will be reflected with the client’s development environment and they will help to determine the final test processes that will be created in this project. The selected changes to the test process should improve the efficiency of the development process by the implementation of better tools for documentation, communication and validation of the user’s needs.

2.1.1 Waterfall model in software testing

Traditionally software development follows the waterfall model that the figure 1 describes. This model dates back to developing concretical products, such as hardware rather than software products. Waterfall model sets a phase for each activity that is needed to complete the software development project. This model suggests the product to be ready before testing takes action. According to the Oxagile's article *Waterfall Software Development Model* (6) test phase mainly includes dynamic black box testing that is targeted to find bugs in a working system. Similarly to the client's current test process, the Waterfall model's testing phase does not focus on validation of the product nor to the user's perspective. The discovered bugs are then reported and fixed before product delivery.

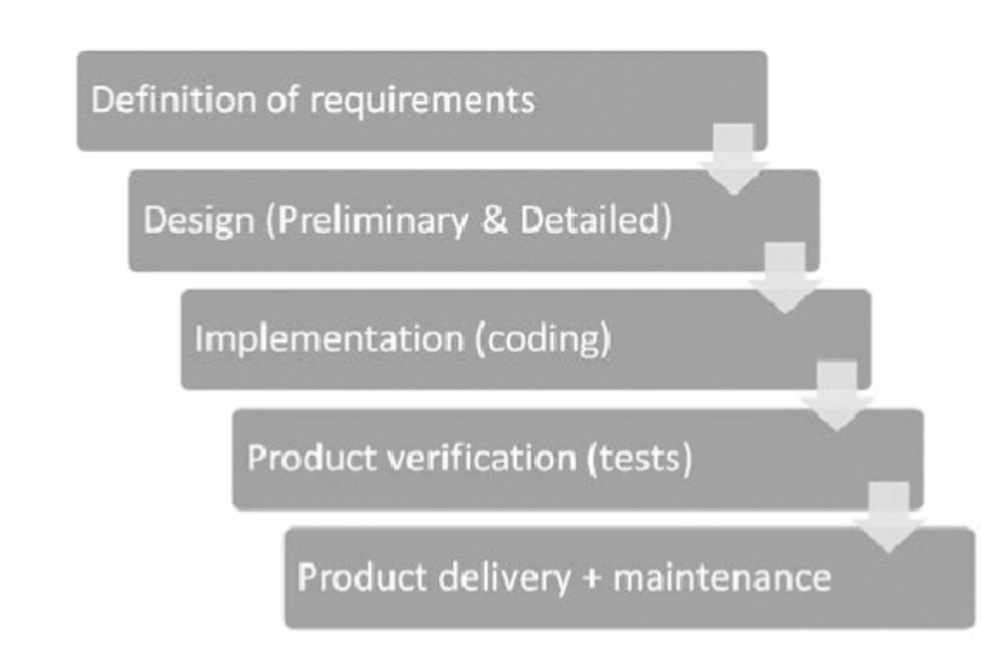

Definition of requirements
Design (Preliminary & Detailed)
Implementation (coding)
Product verification (tests)
Product delivery + maintenance

Figure 1. Waterfall model according to *Fundamentals of software testing* (7).

This model is rather problematic in testing point of view, because the waterfall model does not describe the phase where users execute a static test for the product and find out whether the product actually fixes their problems or not. Moreover, there is a risk that the requirements are not clear enough and the developers misinterpret and end up creating something else than was originally asked for. Mainly this issue is avoided with strict, formal and time consuming documentation. Often in a Waterfall project the documentation and development stages are time consuming because they are so specific. During that time changes for the requirements might come up from the customer’s side. According to *Testing Exellence*’s *Waterfall model*, this kind of changes can break up the entire development process (8) as the requirement specifications in the beginning are fundamental for the entire software architecture.

In waterfall model the software testing process can be extended to the V-model that is displayed by figure 2. V-model is a project management model that is designed to guide the production team throughout the entire development process from the idea to the finished product from the test management point of view. Unlike the figure 1 and Waterfall model suggest, in V-model testing also includes static testing in the beginning of the process when the requirements are discovered and then set. This will be followed by the official testing phase that figure 1 describes too; dynamic testing in the very end of the development process. This phase is inspecting weather or not the product is what was asked for and all the requirements are met. But just like the waterfall model, V-model does not provide much detail nor tools to what happens in the depths of the V: in the production. This is what V-model gets criticized the most about.

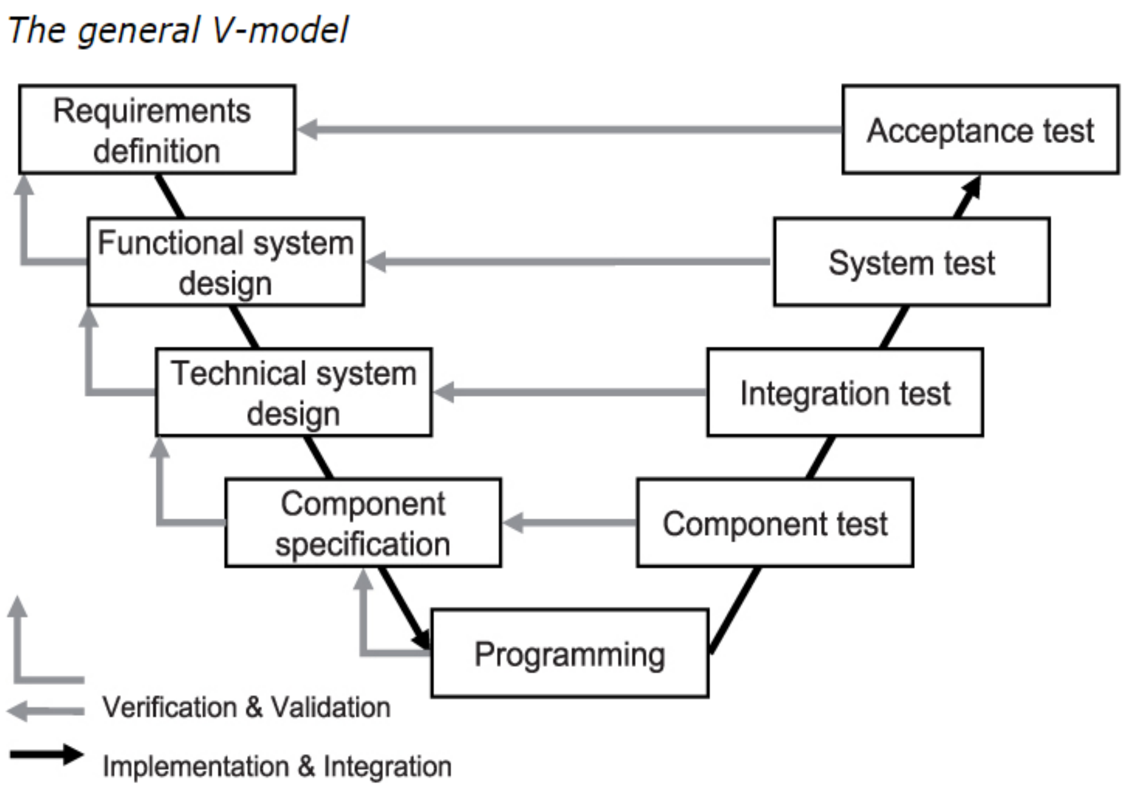


Figure 2. The general V-model according to *Software Testing Foundations* (9).

V-model introduces an enormous improvement to the closed and strict dynamics of the waterfall model. It brings conversation and mutual understanding between the customer and the developer as well as including the user to the production process. Even after all these efforts the user's needs are only taken in consideration in the beginning and therefore changes in circumstances are hard to implement. V-model is also strictly conservative about test documentation which can be beneficial in legal situations, when the product is massive and collects private data (etc. bank application). Creating test documentation is time consuming and therefore it is not recommendable for smaller projects in the scale that V-model introduces it.

The client of this project will definitely be able to benefit from the ideology behind the V-model, which happens to be very similar to the ideology of Scrum development model. Importance of user's impact, communication, verification and validation are all there, only the cycle length and amount of documentation are vastly different. Implementation of better tools for documentation, communication and verification of the user's needs should improve the efficiency of the development process. Static usability testing is highly important way for finding and defining the user's needs and it should definitely be implemented as a part of the test processes in the client company.

2.1.2 Agile testing

Agile software development is based on development cycles and continuous improvement. According to *Software Testing Foundations* Agile production process is all about including the customer to the production process and this means including them also in the testing in an early stage of product development (9). Frequent tests performed by the customer and user make sure that what the programmers and designers are building exactly what is needed. These tests could and, in most cases, should be performed also as static testing. The reality of the matter is that most customers are more likely to be capable of expressing their wants and needs specifically and directly with a touching surface to the first glimpse of what is being created.

QA Lab illustrates their Agile test process as displayed in figure 3 (11). It is an accurate and detailed description of Agile development process and how testing is taken in consideration in all phases but mainly executed in the "software build" sections. The visualized methodology and way of thinking about testing as an entity would be possible to customize to serve the client's needs. This workflow is already quite descriptive of how the development projects work, so it can be implemented with the current resources. More about resources and implementation

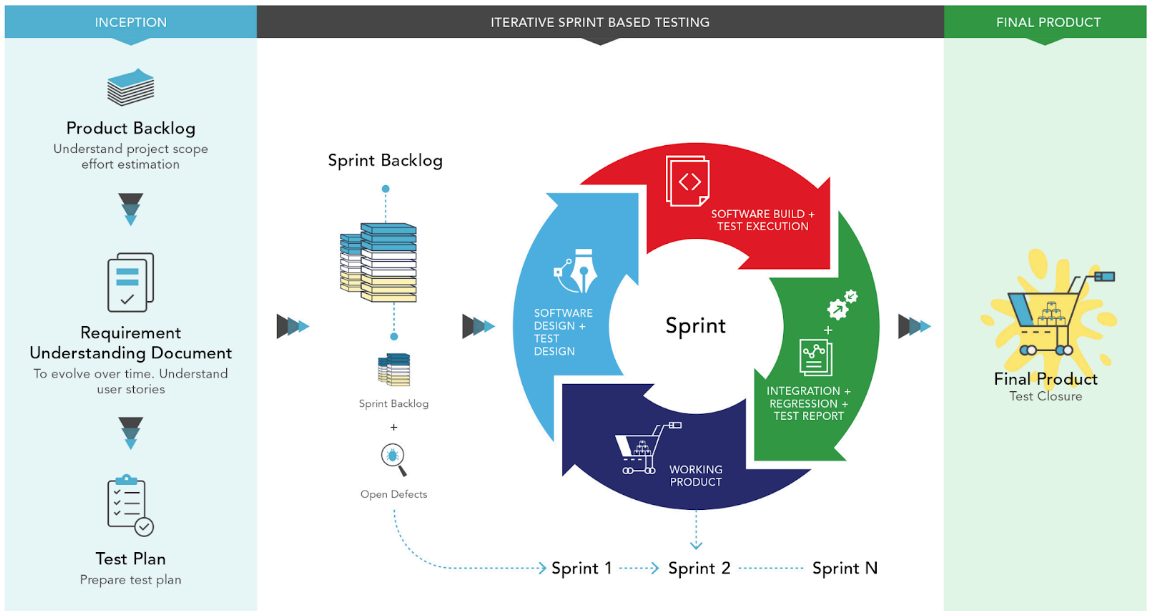

INCEPTION
ITERATIVE SPRINT BASED TESTING
FINAL PRODUCT
Product Backlog
Understand project scope effort estimation
Requirement Understanding Document
To evolve over time. Understand user stories
Test Plan
Prepare test plan
Sprint Backlog
Sprint Backlog
+
Open Defects
SOFTWARE BUILD + TEST EXECUTION
SOFTWARE DESIGN + TEST DESIGN
Sprint
INTEGRATION + REGRESSION + TEST REPORT
WORKING PRODUCT
Sprint 1
Sprint 2
Sprint N
Final Product
Test Closure

Figure 3. Description of Agile testing process (11).

On the left hand side, in the “inception” section, figure 3 states that user stories and requirements need to be realized and test plan should be created based on these. For this process model it is important to provide tools for the client to be able to create a user story, requirement understanding document and test plan. All of these are considerably new concepts for the client, but necessary to get familiar with. We feel that the customers and users are interested in the product quality and will be available for consulting about the user stories and requirements. If they are not willing to work together for some reason, the development team will arrange user experience testing internally and create user stories and requirements, that can be validated by the customer afterwards.

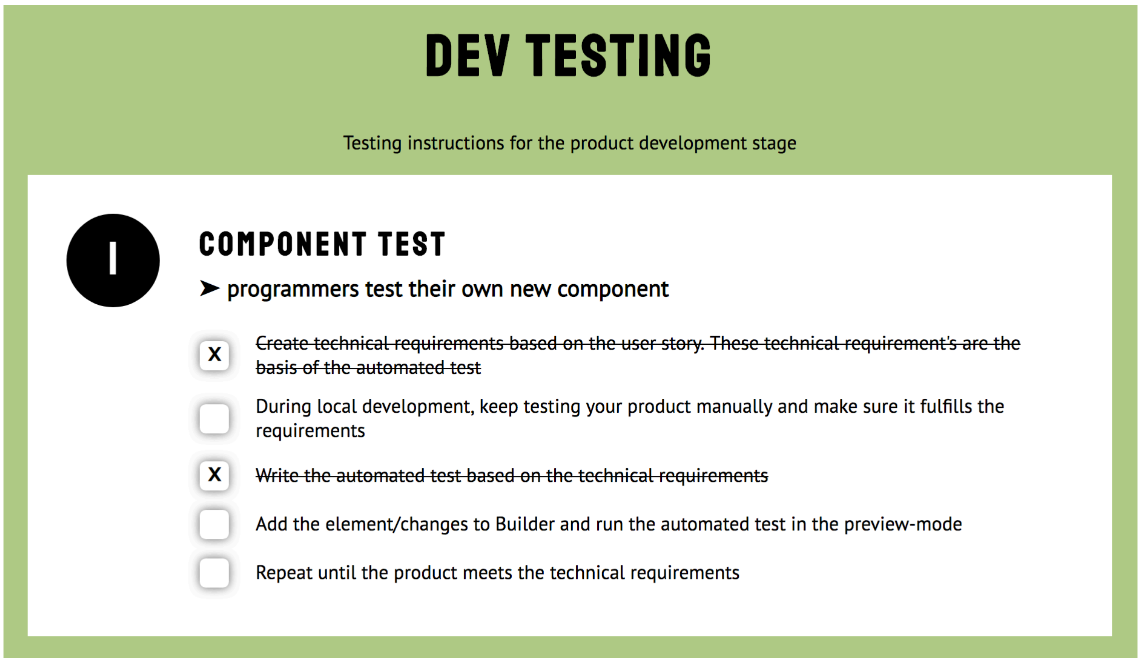
By executing the tests in production and detecting the mistakes and bugs in the development phase they can be fixed more time and cost efficiently. This requires continuous, committed testing throughout the product development. Test driven programming simply means writing the automated test before or during programming the function to be tested. According to *Software Testing Foundations* (9) this method allows testers to work in a way that the test in fact, gets tested by the program instead of the traditional way of programming a function and then testing it out. The process of writing out the test before the function allows the programmer to have a moment to determine what the requirements for the new function are in practice.

2.2 Standardizing the client’s test processes

To implement test automation to the test environment it is necessary to create structure for the test processes. Creating a test process that is clear, where everyone has a specific role and that is easy to execute as a routine practice is very important to run the testing rapidly. The current test process does not define roles or rutines very cleary in the client company at the moment. This chapter looks into tailored tools for updating the client’s test process.

There are a few general test cases documented in the Test Document (Appendix 1). The instructions stated in the test document are more like reminders for general testing, not really directing the tester. As seen in Appendix 1 some of the directions are not even tests. Most of them do not help the tester, because they are not telling what the desired outcome of the test actually is. The documentation of requirements for the test objects are usually completely missing. The requirements can only be accessed by directly asking the account manager, customer or a development team member who has knowledge of the requirements. The Test Document (Appendix 1) is also targeted to the software developers. Referring to Montvelisky’s article *Why developers are poor testers and what can be done about it* (12), developers are not supposed to be the only testers especially not when it comes to their self written program. Developers should be sure that what they have made works, but they can not be the people performing all tests and especially not the acceptance tests.

We decided to create simple and quick processes in a check list form to guide different kinds of testers within variety of test cases. Figure 4 displays a component test with the most basic testing steps in development component test case. It also demonstrates how the steps can be crossed over. We collected these test process instructions into a new website called Test Docs. The core purpose of this website is to give clear instructions to the testers in a place that is easy to find and use. Figure 4 is a screen capture of Test Docs. It also going to includes other results of this research project, such as general instructions for test automation, installation, documentation of requirements and basics of test theory. The website was slightly inspired by Test Document (Appendix 1) and other undocumented test processes that were already used by the client. The previous test processes affected the categorizing of the test cases.



DEV TESTING

Testing instructions for the product development stage

I

COMPONENT TEST

➤ programmers test their own new component

- [x] ~~Create technical requirements based on the user story. These technical requirement's are the basis of the automated test~~
- [ ] During local development, keep testing your product manually and make sure it fulfills the requirements
- [x] ~~Write the automated test based on the technical requirements~~
- [ ] Add the element/changes to Builder and run the automated test in the preview-mode
- [ ] Repeat until the product meets the technical requirements

Figure 4. Sample of a component test task list from *Test Docs* website.

Second motive for building this website is to instruct the client how to implement and use test automation to make these processes easier, faster and nicer to execute. There is also a page for the test developers and test specialists that includes all documentation of test automation with the tools used in this project. In addition to installation options and intructions the page includes tips and best practices for when the automated tests have to be updated or a new test has to be created. It also introduces the minor changes that were made to the production environment structure in order to make the test automation work in desired way.

Third motive is to help the development team members to understand the test results of automated tests and react to them. Test automation does not really fix anything on its own, so it is important to get the test data out and then direct it to someone who can respond to it by fixing the bug. Read moreore about this topic in *4.3 reports and reactions*.

2.3 About test automation

Test automation is a beneficial asset to perform continuous and fast testing within a reasonable budget. Meyers, Sandler and Badgett stand in their book *The Art of Softwa-*

re Testing (10) that automated testing is more reliable than manual testing approaches. Manual testing is seen as necessary for certain problems, but also as a threat for the targeted production time as “it may itself introduce bugs”. Therefore it can be stated that test automation improves the efficiency and repeatability of testing in unbiased perspective. Test automation is popular, not only for bringing down the testing costs by working fast and with minimal human supervision, but also for improving the uptime of the service by detecting the bugs early and fast. Test automation data can be also used to improve the service uptime, since it helps to find the software environment that was still working and that information can be used to detect the possible causes of the bug.

More specifically the client is interested in GUI testing that stands for Graphical User Interface testing. GUI testing focuses on how the software is performing in the user interface level or as a whole, not really taken deeper level functionality in consideration. Fundamentally it checks that the application is running, and does not care *how* it is running. Partly it is chosen because it is realistic to execute in the limitations of the budget and resources that are available for this project, but it also offers the kind of stabilizing functionality the client is looking for.

Test automation and GUI testing are good solutions for the client, because the client’s business model includes technical support of hundreds more or less similarly structured e-learning modules. The functionalities of these modules are based on the base software service’s features that is used for building the e-learning modules like the figure 7 describes. When there is a system update to the platform all these modules have to be inspected carefully, since some of them could have fatal impacts caused by the update and the client naturally has to ensure steady service level to its customers.

2.4 Choosing the test automation framework and test environment

The client firm is rather small and does not want to invest in an expensive test automation solution. Looking for open source alternatives for test automation there are not much options, since it is a competitive business at the moment. The client needs a tool that is easy to use and maintain, even for people who are not programmers for their profession. Even if the test cases has to be written and edited by programmers, it should be possible to execute the tests for a non-programmer. Also it is important to receive readable reports in a pleasant form that does not feel overwhelming like for example a continuous stream of emails might feel.

Robot Framework has a very good reputation as a free, powerful and flexible test automation tool and it fulfills the customer's needs. It is built on Selenium and based on Python, but the user will use Robot Framework's own domain specific language (DSL) for writing tests. This DSL is experienced either as convenient or confusing depending on the user's preferences.

The biggest criticism against Robot Framework is that it usually requires a complicated installing process. This problem has been tackled with RobotLab package that is very easy to install, but it is new and less known option and it has a very small user group. It is also developed as an open source project by two men. This means that there is no guarantee for continuous support for far into the future, since they are developing RobotLab non commercially.

If you want to set up Robot Framework without RobotLab package, you have to take in consideration a long list of applications, tools and settings on your computer. This will take an hour or two, but once the process is completed, it does not need much maintenance. The benefit of manual installation process is that the user has more power over the extensions and versions selected. Although, if the user is a developer who needs specific versions of these dependencies elsewhere, the user has to remember to keep those versions controlled. This issue can be solved with using a virtual machine. Virtual machine is a virtual computer system that is running on top of another computer system, Chris Hoffman describes in his article *Beginner Geek: How to Create and Use Virtual Machines* (13). Virtual machines are good for keeping different environments isolated, in this case test environment separate from development environment. It might be a lengthy process to set up a virtual machine depending if the user is already familiar with or not.

Other viable option for test automation is Katalon Studio that is also based on selenium. Among other conveniences it offers a simple setup package with browser plug in for recording actions. It is easier to use for people who are not comfortable with programming and other work that demands IT knowledge (like using command prompt).

Katalon Studio works quite well for what it is supposed to do, but it is advertised for no programming test automation tool and that is not true. The user is capable to produce a very simple test with a recording feature. However, if the test case is even a bit more complex or needs variables or modular structure, the user gets forced into a Java/Groovy programming environment that is not necessarily bad, just not what the user might predict. Any tester could enjoy testing with Katalon, if they enjoy programming tests with Java/Groovy or find the recording feature to be enough for the complexity of their tests.

As demonstrated in table 1 Katalon Studio offers basic features for a tester who does not have much knowledge or interest in programming, or in the other end of the spectrum it offers remarkable freedom for a tester that really wants to program tests with Java. Robot Framework could be seen as some kind of a midway; it does not have the most easy to use recording tools that work entirely without code, but also you can use it with a more readable domain-specific language (DSL), which is so easy to learn that you do not need a programmer background to master it quickly. It is a close comparison between two constantly improving test automation tools.

Table 1. Comparison of Katalon Studio and Robot Framework.

Feature	Katalon Studio	Robot Framework
Built on Selenium	Yes	Yes
Free and open source	Yes	Yes
Easy install	Yes	Yes
Recording tests (without code)	Yes	No
Writing tests with real programming language	Yes	No
Writing tests with simple code with DSL	No	Yes
Readable test cases for non-programmer	Depends	Yes
Variety of clear reports	Yes	Yes
Feeding data	Yes	Yes
Custom keywords	Yes	Yes
Modular approach to testing	No	Yes
Total	**8½**	**9**

We decided to use Robot Framework because it is slightly more powerful than Katalon Studio and the test cases are written in a clearer format. It also offers a modular approach for creating tests and that is convenient for the client. The client does not need action recording functionality or Java programming, as the same test cases can be written in Robot Framework's DSL in a more readable way. This readability of test cases supports the earlier discovered ideology of tests being part of the communication between developers (testers) and management.

Robot Framework offers possibilities to run tests with different softwares and devices. Our client creates web software and therefore testing will naturally happen in a selection of different browsers. By using different Selenium drivers it is possible to run same tests on different browsers and find out if there are any browser specific bugs. Browser testing is one of the most automatable test cases in the client company and it is very useful to taggle in this project. This is not really Robot Framework specific possibility, but definitely a must-have.

As we decided to use Robot Framework new research question came up; what is the most efficient way to use Robot Framework for creating and maintaining tests in the client company? By researching the websites and recent events of Robot Framework we found a remarkably useful document *Robots from Jupyter* (15) by Asko Soukka and Nicholas Bollweg. It gives detailed information about writing tests with Selenium Library and Robot Framework in JupyterLab development environment, that is basically a visual interface for developing anything and everything. They also created a package called *RobotLab* (15) that includes all the tools needed for developing and deploying tests instantly from installing forwards.

RobotLab is very easy to install and use. One of the perks is that the development environment is quite well controlled so that it is easy to share with other developers and the program will work in any local environment. JupyterLab projects are quite easy to share and therefore the tests created with it can be easily deployed by anyone - even a person who never ran a test with Robot Framework before. It also offers a convenient tool that helps the user to find and use correctly the desired Robot keywords. Especially for a new user it is very beneficial to instantly see what variables each keyword expects. The only downside to this is the inconvenient package size (400-500MB). It is of course possible to download JupyterLab and Robot Framework separately, but then the user is required to configure JupyterLab manually.

2.5 Test documentation

Starting point for the client's test documentation is that there is not very much test documentation. Tests are based on subjective assumptions of how things are supposed to work. Specific test cases or expected test results are never written out. At the most, they are spoken out loud and communicated from the management to the testers, but often the test environment does not allow the testers to communicate directly with the customer. This is why the management often hands the ball to the customer to execute the acceptance testing. There should be documentation of what the requirements were in the beginning of the project and what the test results should be at that point, but that documentation rarely finds it way to the testers. These documents are emails and they are not very descriptive or useful for testing purposes. Mainly this entire process is based on verbal communication and individuals' memories and perspectives.

Figure 3 offers a clear solution to the pre-production part of the documentation process and quite a remarkable amount of steps for executing and documenting tests throughout the entire project. By studying the figure we came up with a solution to the inception level documentation.

One convenient part of Robot Framework is that it automatically creates test reports in HTML form that is easy to read, share and save for later if needed. Major part of the test automation process is writing out the user stories and translating them into test cases that are written in Robot Framework's DSL syntax.

Best way to approach this problem as a whole, would be starting to write out simple test cases. This would benefit the client especially when performed in the beginning of the test process. Even instructions for the developers could be formatted in a test case form, that explains what the user is doing, what the program is doing and what is the expected outcome. Documenting the requirements in this way would be very efficient and radically reduce confusion and time when it comes to both; understanding the goal of the project with minimum requirements and acceptance testing.

This kinds of changes take a lot of commitment from the management. Therefore an essential step is to provide the management with tools to create user stories and requirements with the customers. These user stories can then be inspected and evaluated within the client company's development team and detailed requirements and test cases can be written out based on them. This will offer another step to deliver these requirements and test cases to the customer and make sure the development teams are on the same page with the customer.

3 Implementing test automation

The goal of this chapter of the research is successful implementation of test automation in the client company's test processes. This chapter takes a deeper look into finding the most efficient way to benefit from automated testing. The tools for test automation have already been declared, but here are several big steps that needed to be taken in consideration. First is setting up the test automation environment. This part is simple and straightforward, but still has to be done correctly. When JupyterLab and Robot Framework are running, it is time to dive deeper into Robot framework and modular testing. According to Amy Reichert's article *Software testing for complex, integrated applications? Go modular!* modular testing is a "way to break down application functionality into small pieces" and then reuse these pieces accordingly to the specific test cases (16). Fortunately Robot Framework is built in a way that supports modular testing very well.

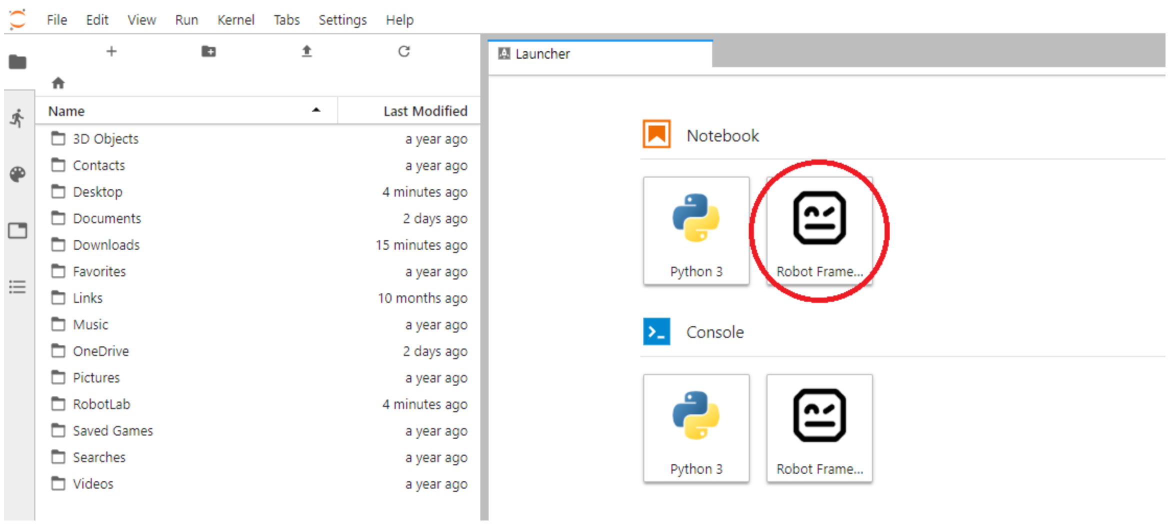
With the knowledge gathered from these parts of the research it will be time to take a deeper look into test cases that are specific for the client's needs. First part of the case research is automating test processes in the software application, that is used for building e-learning modules. Following comes the modular testing of a basic e-learning module. That chapter looks into how modular testing can be executed with Robot Framework.

Finally we look into Reports and reactions alongside with documentation and maintainability. Single automatic tests created in this research are not very useful on their own, if they can not be shared and maintained. Therefore it is important to put effort in the maintainability and documentation and give tools for the testers to keep their tests up to date with the software to be tested.

3.1 Setting up test automation environment

After trying up few different setups of Robot Framework combined with variety of GUI (graphical user interface) tools it turned out that there was a much easier way to do it. By installing Robotlab, the package mentioned earlier, the user can really focus on the testing part and not a complicated set up. This is especially good if the tester is not familiar with programming and updating advanced user settings for their operating system. GUI also provides testers easier ways to write and execute tests and view the test logs and screen captures.

The user will start their effortless RobotLab installation process by visiting the following URL: https://github.com/robots-from-jupyter/robotlab/releases (14) and choosing the version that suits your operating system. Install it by following the instructions of the installer, installing may take quite a while due to the inconveniently large size of the package. After installation you can launch RobotLab by double clicking the desktop icon. Unlike most software, RobotLab opens a new terminal/command prompt window that displays the link to your JupyterLab page. It also opens the html file in the software that your computer is set to automatically open html-files in. For example in a browser or a code editor. If it does not open in a browser, copy the link from the terminal and paste it to your preferred browser. After this you should be viewing something like figure 5 presents.



Name	Last Modified
3D Objects	a year ago
Contacts	a year ago
Desktop	4 minutes ago
Documents	2 days ago
Downloads	15 minutes ago
Favorites	a year ago
Links	10 months ago
Music	a year ago
OneDrive	2 days ago
Pictures	a year ago
RobotLab	4 minutes ago
Saved Games	a year ago
Searches	a year ago
Videos	a year ago

Figure 5. JupyterLab/RobotLab opening view

Robot Framework extension in Jupyter Notebook can be opened by clicking the icon marked in figure 5 with red circle. The user will be instantly directed to the test suite that contains by default one empty cell. To activate Selenium Library, user should select the first cell and at the code from Listing 1. Then this cell needs to be run by clicking the play button that is located in the top of the window. Now Selenium Library and Robot Framework are set and ready so the user can start adding more cells and creating first test cases.

```
*** Settings ***

Library  SeleniumLibrary
Suite teardown  Close all browsers
```

Listing 1. Settings that are required to be run before any selenium keywords can be used in tests.

A simple test case, written with Robot Framework, contains a name that is followed by the keywords listed below. Listing 2 displays the basic syntax: selenium keyword is followed by the variable(s), and these two have to be separated with two or more spaces. This is then modified according to the functionality that each test case demands.

```
*** Tasks ***

Open browser
Set Selenium Speed  1 seconds
Open Browser  https://www.google.fi/
Capture page screenshot
```

Listing 2. Simple test that opens a browser and captures a screenshot.

You can find all the available keywords and specific functionalities from Robot Framework user guide (20), SeleniumLibrary documentation (21) and BuiltIn documentation (22). These documents are quite large due to the large variety of keywords. If you have experience on Python, remember that many of the keywords are based on Python keywords. It can ease the process of getting familiar with the DSL.

3.2 Smart test automation for evolving web software

Like one might come to expect, it is relatively simple to create an automated process for a static environment. The challenge is to take in consideration how the test will stay or be kept functional when the software evolves. The client's development process can be adjusted to compliment the previously created test automation format; similarly as test driven programming that was mentioned earlier. Even while doing the best and most versatile tests, they have to be adjusted if there are major changes to the application. This chapter is about finding the best methods and processes for creating flexible and modular test automation solutions.

The best of Robot Framework will be brought out when the tester begins to write their own keywords, which are essentially lists of case specific actions created with Selenium keywords. Robot Framework also encourages the user to create data-driven tests and using variables for test data. This works towards tests' reusability and makes it easier to run multiple tests with same data. These features allow and encourage modular testing systems, which are exactly what the client needs for testing their products by the quickest and the most convenient way possible.

Fortunately the most common test environment and the focus of this research project, regular e-learning module, is already quite clearly sectioned. Like figure 6 displays the client's test processes will be targeted to compact and strictly scoped sections and therefore the tests can be limited to the scopes of these components. Traditionally tests need to be carried out in order like *case specific testing* explains. Modular testing is targeting a situation that is very ideal from the client's point of view: the order of the components is trivial as the modular testing system is capable of detecting the component type and running the test case assigned to that component.

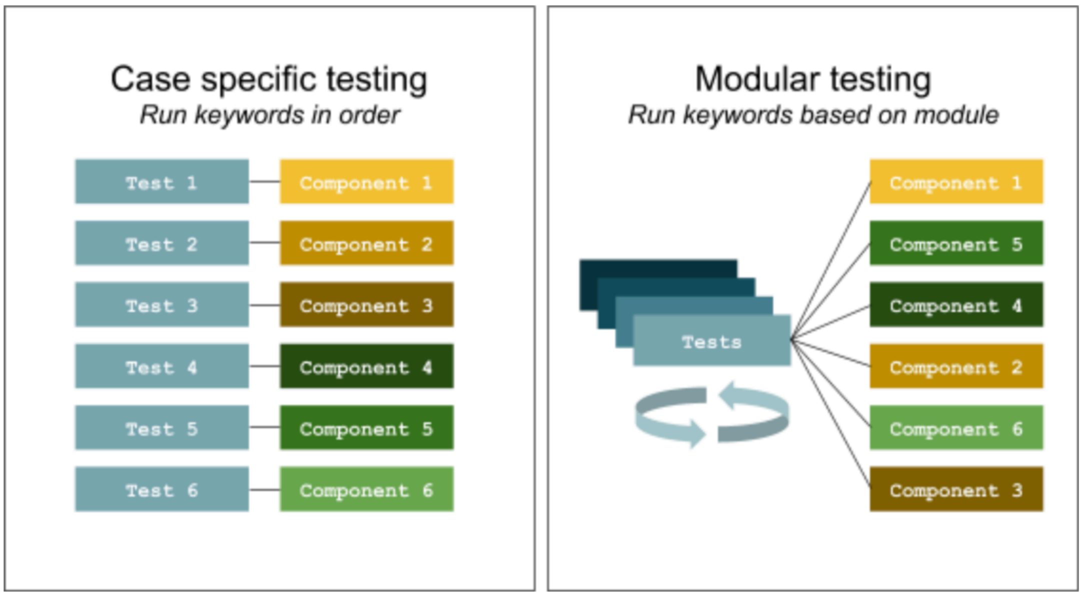


Figure 6. Visualizing the impact of modular testing.

The modular system is also good in the sense that the tested e-learning module does not have to "contain" all the test cases, and this is remarcable from the version control point of view. What this actually means is that the test can be edited and updated, if another project for example has all other same components but one extra. Then the original test case can stay the same but one new keyword and module is added to the test case. The updated test case can now be used to all the previous e-learning modules and the new one too.

3.3 Test automation in the client's software platform service environment

Occasionally the client also needs to test the software platform, that is used for building these e-learning modules from the GUI perspective. One of the reasons why updating the software is a hard and timely process is that the testing of these updates is highly time consuming and therefore expensive. Automating simple tests can open doors for continuous delivery and smaller updates in shorter development cycles.

Figure 7 illustrates the software architecture of the client's platform for building tailored e-learning solutions, that is the fundamental product of the client's business. The architecture seems logical in the picture, but the technical realisation is quite surprising in the sense, that the GUI is built on many levels of different websites that are just nested inside one another to look like a unified, complete web application. This is quite frustrating from the testing point of view, but not unconquerable. Selenium and Robot Framework have taken frame structure in consideration and it is possible to work with this issue.

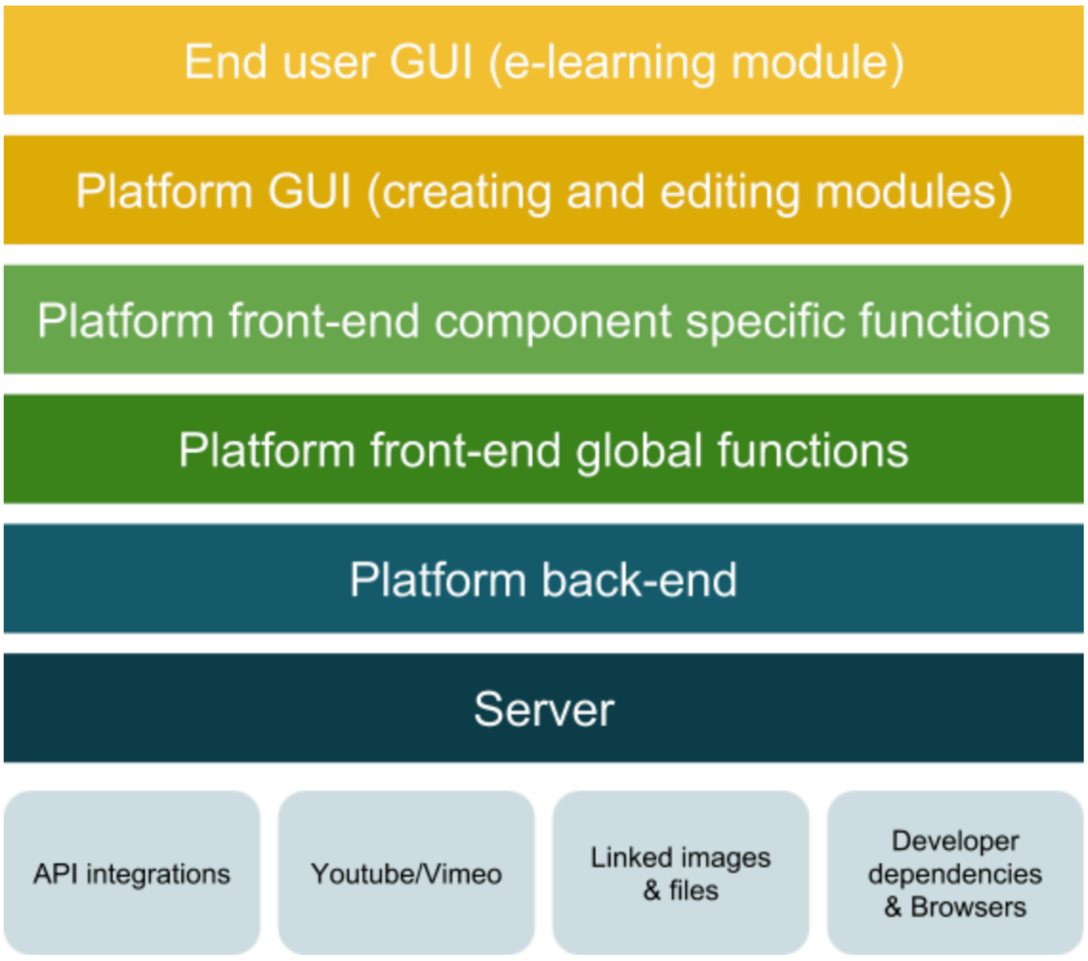

End user GUI (e-learning module)
Platform GUI (creating and editing modules)
Platform front-end component specific functions
Platform front-end global functions
Platform back-end
Server
API integrations
Youtube/Vimeo
Linked images & files
Developer dependencies & Browsers

Figure 7. Client's software structure and all the evolving components.

One problem down, let's tackle the next. The platform has a lot of moving elements, that do not have unique identifiers. This causes excessive use of xpath for recognizing elements and therefore it is close to impossible to create reusable test cases that could be run whenever without human interaction directly with the application.

Modular testing would be the solution for this testing purpose too, but during the study it turned out that it is not really possible to carry out successfully within the time restrictions since the software is working in multiple frames and levels and the structure is not very consistent and the functionality is not fully repeatable. Some of the test cases can be reused, but they have to be edited to work as desired. Listing 3 is an example of how the software structure limits the possibilities to reuse the test cases by displaying the parts that need to be manually edited for reusing the test case.

```
Open intro element
    Click element  xpath://div[@id="container"]/ul/li[1]
    Select Frame  class:iframe-edit
```

Listing 3. Example of multiframe testing. We are using the keyword "Select Frame" to move between nested websites.

Another less significant problem is that there is also a layered structure in the frontend that can provide the tester with a lot difficulty, in both normal use and automated testing. If a facilitator (user) wants to edit a picture they have to hover over the image and carefully avoid the elements that are layered on top of the image and move the cursor over a small icon in the top left corner of the image. This is very hard to do manually too, but with automated testing it is incredibly frustrating. Listing 4 explains how it is still possible to try to avoid the text boxes that are layered on top of the image.

```
Edit element image
    Select Frame  class:iframe-edit
    Click Element At Coordinates  class:image-wrapper  -100  -110
    Click element  class:imageEditorOpener
    Unselect Frame
    Click element  xpath://div[@id="img-dialog"]/div[1]/a
    Select Frame  id:img-library-iframe
    Click element  class:lib-image
    Unselect Frame
    Click element  class:insBtn
```

Listing 4. Test case of editing background image

Because of the inconsistent multi level environment it is not really going to be worth it to recommend GUI testing in the building platform environment as a routine tool for most common front-end development related tasks. GUI testing can however be useful for running specific test cases, especially when testing is related to system updates of the platform instead of a specific e-learning module.

Other slightly further fetched use case could be robotic process automation (RPA). Currently there are a lot of repetitive tasks that should be automated to save everyone's time and nerves, but that is not really what Robot Framework is made for and more importantly it is out of this project's scope. This subject is discussed more specifically in chapter *5.2 Ideas for further development*.

3.4 Test automation in the end user environment

One of the main objectives for this research project was to create a test case that applies to all e-learning modules that are based on the most recent module template. The aim is that this test case can be used to test large bulks of e-learning modules, as it is so well modularized that it can be used for testing any module based on this module template without editing. This specific test case will set the example to the tests that will be created in the future and also be ready-to-use option for testers that are just getting to know Robot Framework for the first time.

This automated test for all modules based on the template will have several use cases. It can be used to test specific components in the development or system integration test phases. It can be used for browser testing, or testing with different devices along side with testing the responsive aspects of the GUI, such as scaling and accessibility with all viewport sizes. With the test driven development mindset the automated test can even be used as an acceptance test to a certain degree, as the component specific tests include minimum requirements of those components in themselves.

In production it can be used to test a large amount of modules, when there is a system update to the platform that is running in the background. Figure 7 also displays other dependencies, such as external video players and browser features that may have had changes in them since deployment and therefore it is good to run the test every now and then especially if there is a known change to the dependencies. It is highly important to stay aware and updated about these moving factors that affect the GUI sometimes even fatally.

This test is structured in a reusable and modularized way that the figure 6 suggests. Component specific tests are nested in compact, clearly named keywords, that are built based on Selenium keywords and custom keywords that are created to make the most repeatable tasks easy to use. These first-level custom keywords are for example clicking a button to move to the next component or testing if it is possible to move to the next component, when it should not be possible.

A good component test includes tests for positive and negative cases, takes all user stories in consideration and is understandable for a team member, who has not written it. Listing 5 describes a basic login component keyword, that is part of the created test. It was chosen as an example since it has a large variety of different positive and negative tests and a lot of user stories compared to other more simpler components.

```
*** Variables ***
${FIRSTNAME}  test_fname
${LASTNAME}  test_lname

*** Keywords ***
Click ForwardBtn
    Click element  class:ForwardBtn  #Should be possible to continue

Click LoginForwardBtn
    Click element  class:checkLoginBtn

No Click ForwardBtn
    Run Keyword And Expect Error  Click ForwardBtn  #Should not be possible

Test login
    No Click ForwardBtn  #Should not be possible to continue
    Click LoginForwardBtn  #All error messages should be visible
    Element Should Be Visible  class:firstname-error
    Element Should Be Visible  class:lastname-error
    Element Should Be Visible  class:dobmonth-error
    Element Should Be Visible  class:dobyear-error
    Input text  id:fname  ${FIRSTNAME}
    No Click ForwardBtn
    Click LoginForwardBtn  #Other messages should be visible except firstname
    Element Should Not Be Visible  class:firstname-error
    Element Should Be Visible  class:lastname-error
    Element Should Be Visible  class:dobmonth-error
    Element Should Be Visible  class:dobyear-error
    Input text  id:lname  ${LASTNAME}
    No Click ForwardBtn
    Click LoginForwardBtn  #DOB messages should be visible others should not
    Element Should Not Be Visible  class:firstname-error
    Element Should Not Be Visible  class:lastname-error
    Element Should Be Visible  class:dobmonth-error
    Element Should Be Visible  class:dobyear-error
    Select From List By Value  id:dob-month  05
    No Click ForwardBtn
    Click LoginForwardBtn  #DOB year message should be visible others not
    Element Should Not Be Visible  class:firstname-error
    Element Should Not Be Visible  class:lastname-error
    Element Should Not Be Visible  class:dobmonth-error
    Element Should Be Visible  class:dobyear-error
    Select From List By Value  id:dob-year  2001
    Click LoginForwardBtn  #All error messages hidden, ForwardBtn visible
    Element Should Not Be Visible  class:firstname-error
    Element Should Not Be Visible  class:lastname-error
    Element Should Not Be Visible  class:dobmonth-error
    Element Should Not Be Visible  class:dobyear-error
    Element Should Be Visible  class:ForwardBtn
    Click ForwardBtn
```

Listing 5. Example of a component test of login component

The main test includes a loop that inspects what component is currently visible and runs a keyword assigned to that component. Recognizing the components turned out to be harder than what was expected, partly because the programmers had never had to think about test automation before in their work and had not added unique attributes to the components. Modular GUI testing is not yet as familiar for most developers as it should be. The takeover from this conclusion was to create more materials and tools for developers to enhance their skills and knowledge in this field. These tools include test examples, their introductions and documentation that can help a person who is using Robot Framework for the very first time.

Since most of the components do not include a unique HTML attribute to inspect, there are couple of options to approach this situation. First option is to create these unique HTML attributes to clarify the structure for the test automation purposes. Other one was to inspect the styling via Cascading Style Sheets (CSS), but this is a rather unpleasant way to inspect elements with Robot Framework and in general, since CSS is not very readable nor appealing, especially to the non-technical people. Third option suggested was inspecting elements by the JavaScript function calls that are specific for the component, but this is close to as unreliable as the last mentioned CSS-recognition way to detect components. Some components use same functions with each other, but still have differences in the GUI or other differences such as using multiple global functions.

Problem with CSS recognition is that different customers have different layouts and therefore the CSS-styling is varying a lot between different e-learning modules. Similarly to this it would be quite restrictive for the developers to not be able to change JavaScript functions names or structures without editing the test. These challenges led to a decision that each component of the module had to be assigned a global, recognizable class-attribute, that is same for each component and unique id-attribute. The new unique id-attribute will perform as a dual feature since it is also useful in another structural task: determining the component type. Component type is used for calling component specific functions within the building platform. Currently the component type is not documented very efficiently either.

Listing 6 is an example of the main test. Variable ${ELEMENT_TYPE} is first defined in the *Variables* shell. The main loop is detecting weather or not ${ELEMENT_TYPE} is *finish*. Every e-learning module has a finish-component as the last component of the module. This way it is possible to exit the loop when the module ends. Within the loop we assign the ${ELEMENT_TYPE} as the unique id-attribute of the visible component. Then follows a long list of component checks, and by comparing the ${ELEMENT_TYPE} variable to given page types the test selects which keyword has to be performed.

```
*** Variables ***
${ELEMENT_TYPE}

*** Tasks ***
Check element
     Set Selenium Speed  1 seconds
     Exit For Loop If  '${ELEMENT_TYPE}' == 'finish'
          ${ELEMENT_TYPE}=  Get Element Attribute  class:testable-element  id
          Run keyword if  '${ELEMENT_TYPE}' == 'intro'  Test intro
          Run keyword if  '${ELEMENT_TYPE}' == 'login'  Test login
          Run keyword if  '${ELEMENT_TYPE}' == 'finish'  Test finish
```

Listing 6. Keywords are looped through and components get detected

The id-attribute is also handy for version control. For example if there’s a new version of login component that has new features and requirements, the same test can be used but a new keyword with the name *Test login 1.1* can be added. Appropriate id for this component would be *login1.1*. Then this new case will be added to the loop and even if the module has both of these login versions it can execute the right keywords in right contexts.

3.5 Reports and reactions (what happens when there is a bug)

Bug can occur via GUI testing either in development or in production. Executing the tests stated in the test plan in the development environment is very important so that bugs in the production can be minimized. All tests assigned in the inception state need to be passed and the documentation in this case is a message related to the development task that the tests have passed, and the product is ready to be published.

However, if a bug appears in production it is reasonable to file a bug report to the HelpDesk. HelpDesk will reassign the bug fix task to a person who's able to fix it. Common situations why there might be a bug in the production is a change in the software platform that the e-learning module is based on. Changes in the other dependencies of the e-learning module, such as videos or images or changes in the browser, are also commonly cause bugs.

3.6 Maintainability and documentation

The automated tests need to be quick and easy to create and maintain. User experience is something to take in consideration when creating these tests and expecting the users to adapt them into their work and maintain them according to system updates.

First step towards creating clear and reusable tests was naming the custom keywords clearly. The aim was that just by reading the name of the keyword, the tester who has a bit of background knowledge of the components of the module could predict what that keyrod is supposed to do. The focus is that it will be intuitive to update a component and then update that keyword to match the update. This way there might be multiple versions of the same test and those versions could be added to the version control that is already practiced for the e-learning module packages.

Well performed version control is an important tool to control and develop the automated tests. The client has selected GitHub as their version control tool. Since we are using JupyterLab (RobotLab) it is quite straight forward to save the Jupyter Notebooks to GitHub. Changes can be updated through regular Git commits or simple drag and drop on the web application. There are multiple ways to work with this subject and one of them is JupyterLab extension called JupyterLab-git (16). This tool allows easier version control with the Jupyter Notebook and extends the user interface with visual version control management tool.

Sometimes test cases can be very specific to one project and in these cases, it would be the easiest to store the test document in the project's version management. Rule of the thumb will be, that if the test is highly based on a generic module test and reusable in another project as is, it should be kept in the version control of that test. However, if the test is not reusable nor important for any other facet (e.g. project manager) in the project, it should be moved to the version control of the project that it correlates with. Borderline cases such as possibility that someone else wants to run the test independently, the Jupyter Notebook should be kept in its own repository. If the tests are kept in the project repository, they can also be exported later manually, if a need for sharing comes up.

In addition to sharing test cases with technical testers through version control in Github, it can be useful to share automated tests to non-technical people so that they are able to get a link to a test case and immediately start running it independently. With Binder tool Jupyter notebooks that are pushed to Github can be turned into interactive notebooks that can be shared, inspected and executed anywhere and with any browser. Test suits packaged with Binder is excellent for sharing tests with other team members, managers or even customers if needed.

4 Results

When inspecting the client's processes, the main issue of the client's test process was found to be the overall lack of documentation and not consulting the customer enough at the beginning of each project or sprint. This led to confusion in development and testing. The tester was often not able to confirm if the product fills the requirements, as the requirements were not clear. Acceptance testing was often performed by the customer, which did not solve the issue of undocumented requirements, as even the customer may have forgotten them or come up with new requirements during the development phase. The research implies that by creating and documenting user stories the customer could express their needs better. These user stories could be used as the basis for user requirements which refers to the automated tests.

The research showed that test automation is quite effortless to implement technically, but it does require very detailed description of requirements. If the automated test is written clearly and preferably with comments, it can serve as the documentation itself. Test automation was definitely seen as an advantage in the research when it came to improve the testing process, as it would enable faster and more structured testing. It turned out that modular technique for testing was very favorable when thinking of the reusability and maintenance of the tests. Creating tests in modular form is way less of an issue than not being able to reuse the same test case for multiple e-learning modules.

The results chapter is a collection of findings and recommendations of the actions the client should take based on the findings. The research question covers three main areas: what the best test process is, how test automation should be implemented and how the test process can be kept sustainable. Therefore, we sectioned the more detailed results in the three following chapters.

4.1 The most practical test process for the client

The final test process will include all members of the project team more dynamically than previously. The main goal is to work together, communicate and document the user stories and requirements in a way that enables reliable acceptance testing performed by other people than the customer. This new process aims to reduce the confusion between management and development when it comes to requirements, to reduce the time the development takes. The target is to deliver only products that were asked for. Other benefits achieved are reduced amount of bugs detected by the customers, which leads to higher customer satisfaction and rise of security and efficiency when it comes to testing and accepting system updates, which leads to faster development cycles.

Figure 8 illustrates the information flow of a software development project with heavy focus on testing. It is based on the Agile software development process, that was previously illustrated in figure 3. In figure 8 the development process is divided in ”requirements”, ”sprint” and ”verification” sections with the base idea, that all these steps are can either be followed just once in a project if it is very small, or repeated multiple times in a larger project. The project team is divided in ”Account Manager & Customer”, ”External tester” and ”Development team”. These are the main role types that need to deliver and receive information with each other to successfully create, test and verify the software product.

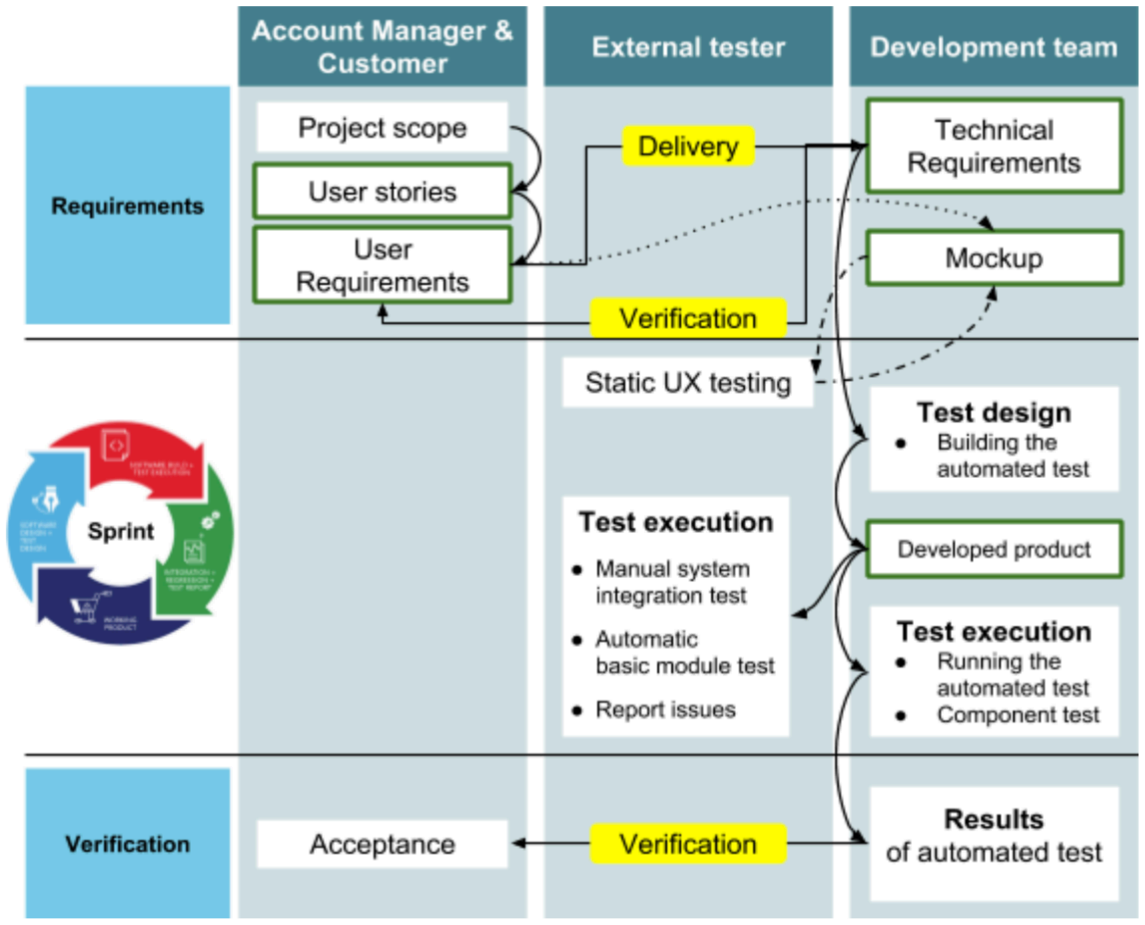


Figure 8. The most practical test process. Deliverables are marked with green borders.

The first one of the main updates to the previous development process are adding user stories as a major tool for the project. This documentation will have a big role as a way for the customer to communicate the user’s needs to the development team. The user stories will work as a base for the development team in creating the technical requirements in documented form, which can then be shown to the customer and verified by them. Technical requirements can be documented either with plain text, swimlane figure or informal drawn description to the process flow. The form of documentation will depend on the preference of the development team member and the scale of the feature or project that the documentation is about. All of this documentation will also serve a major role in testing, because the automated tests will be based on the requirements and, in an extent, on the user stories too. The requirements are also very important for manual tests, as they allow the external tester to have an insight on what the product is supposed to do, as well as what it is not supposed to do. Previous to this project this information was mainly communicated verbally, and sometimes lost for ever with projects that took place a long time ago or the developers of those projects left the company. This change will ease out the work of the client's HelpDesk as they get tools to inspect and verify the bugs found in the software and no longer mistake them to features.

The second main improvement to the development process is the implementation of test automation. The successful execution of the automated test is one of the main requirements for verifying that the product is working. Test automation can not be implemented successfully without having the knowledge of the requirements and that is one of the reasons why it is so important to document and validate the requirements in the inception phase of the project. By implementing test automation the process of verifying the product should get a lot faster and more reliable compared to the previous methodology. By testing with Robot Framework it is easy to collect test data and store accurate information of when the product was working and how exactly it was tested. The test plan is documented very well by writing an automated test and therefore the test it extremely repeatable.

4.2 Impacts of test automation in testing processes

Without a doubt the biggest issue that test automation is solving is testing the large amount of e-learning modules that can now be tested by running just one script. The time it takes to test each e-learning module manually is remarkably greater than the time it takes to run this script. Even counting the time it takes to create and maintain the test suite, it's still more efficient and much more fun than manual testing. The core software, the platform that enables building the e-learning modules, has not been updated as often as what would be ideal. This is due to the fear of breaking dependencies and by that breaking all hundreds of e-learning modules with it. Automated testing offers a great solution for this as when the automated testing is implemented, the changes to the base software can be done more frequently, effortlessly and it is easier to scale up the development team.

The current development hierarchy is that the e-learning module building software is mainly built by an external consulting company and the in-house front-end developers only create the components that are visible for the end user of the e-learning module. In the client's case scaling up the development team means bringing the developers from front-end, deeper into the software side. Test automation will make this process a lot easier, since the manual testing will be non-existent or at least it will have a much smaller role in the future.

It is very useful for the client to be able to use its resources for developing this platform software faster. The customers are expecting better services continuously and it takes a lot of effort to improve and grow in this area. It is also good for the front-end developers to be able to have more responsibility and understanding in the entire software development process. Ideally this change will not only improve the quality of the service but also brings transparency and mutual understanding in the development.

4.3 Sustaining structured testing in Agile environment

First tools for implementing and sustaining structured testing are basic instructions for documenting the initial user stories, user requirements and technical requirements. In addition to these the automated test based on these requirements will work as a more detailed and specific version of the technical requirements. These documents need to be done correctly. User stories are kept generic enough and focused on the users' actual needs. They have to be able to create an entire scenario with the following data: "As a <role or persona>, I want <goal/need> so that <why>" as Nikita Sobolev suggests in his blog (23). User requirements are basically user stories written out more specifically in a statement form and some preferences might be added as details. Technical requirements will be written as the basis of automated test created for this test case, in a similar format as one would write a test in Robot Framework. All of these will be stored in a *REQUIREMENTS.md* file in the project's Github Repository, and can be looked back at while developing, testing or maintaining the project in the far future after the release.

One of the main tools for integration and sustainability was *test-docs website*. For developers or other black box testers it includes instructions for installing RobotLab, test automation examples, user stories and requirements examples and best practices for writing testable code. For all testers it includes instructions for the most common test types, that will ofcourse not serve as well as specific requirements, but they will help to direct the tester in the right direction.

Lastly a very important tool is automated tests version control in Github. There shall be a Github repository created for documenting and controlling the development of the major test suites. If there is an e-learning module that is radically different from the current base module, it is reasonable to open a new repository or include the test suite files to the e-learning module's repository that this test concerns. The structure of the repository will remain clear, when all the tests are listed inside a "tests"-folder.

5 Discussion

In its entirety the research found the following answers. The client's test process should be improved by creating documentation of the requirements in the beginning of the project or sprint. The requirements should work as the basis for test automation. Test automation will be realistic to achieve within the client company, if the requirements for the test subject are well communicated and specific enough. The maintainability will be achieved with trying to make the process as complementary to the previous process as possible. The user stories are a minor, yet efficient addition to the process. Test automation on the other hand is quite major change and it will remain maintained due to the modular way of creating the tests. The tests are reusable due to the modularity and therefore they can be implemented to new projects without much need for changes.

5.1 Pitfalls and ideas for further development

Our research was inspecting the entire development project team, but it was slightly biased from the developer's and tester's view. We can not state the scale of impact this had on the results, but if the research was to be continued, it should involve the management of the client company more.

The research scope was focused on GUI testing and therefore unit testing was left with minimal inspection. Unit testing was found to be an extremely useful tool for the testing process, but it will be more timely when the client's software structure is renewed to more modular form and the technology will be upgraded. Future may introduce tools such as Jest and Enzyme that both are testing tools for React JavaScript framework (12). They can be used for unit testing and enable real test driven programming in a scale that can not be produced with the current development environment. Hopefully the client's development team can start using React and test driven programming in production soon.

Another area for improvement was that there was not enough time in the end of the project to measure the results of the changes that the study suggests. Important measurements for the performance of test automation would be calculating the bug reports from the customers, time that it takes to test new features and time that it takes to fix bugs with and without the suggested test documentation.

Other issue to tackle within the similar field is robotic process automation (RPA) that means automation of common, repetitive processes. It could be used to automate the most common tasks that are annoyingly long processes compared to how often the developers have to execute them in their daily work. For example when React and other more modern and powerful tools are implemented as primary development tools, Jenkins Pipeline could be introduced to ease out these processes as it is easy to automate building, testing, deploying and reporting with this tool. Jenkins Pipeline focuses to continuous delivery which means improvement in synchronizing development team work, and thrives in more rapid development and frequent releases.

5.2 Conclusion

In conclusion the test process should be changed so, that it would involve the customer more in the beginning and less in the end; customer should be consulted in the regard of user stories and requirements and not in acceptance testing. Test automation is a very beneficial tool for the client for GUI testing and acceptance testing. We recommend using the test case that was created in this project for both, development and acceptance testing. How ever, due to the technology that the development is based on, unit testing will not be implemented as a part of this project. The automated test case that was created in this project should remain maintainable because of its modularity. Our approach will direct the testers to create modular and reusable tests in the future and keep the test automation effortless by keeping up that manner. The goal for the final outcome of the implementation of our suggestions is that the test automation will help to create a more structured and meaningful test process for the client.

References

1. Agile manifesto. 2001. https://agilemanifesto.org/
2. SQA. DH3E 35: Software Development: Structured Programming. 2010. https://www.sqa.org.uk/e-learning/SDPL03CD/page_16.htm
3. Guru99. Static Testing vs Dynamic Testing: What's the Difference?. (Read 14.4.2019) https://www.guru99.com/static-dynamic-testing.html
4. Software Testing Fundamentals. Black box testing. http://softwaretestingfundamentals.com/black-box-testing/
5. Software Testing Fundamentals. White box testing. http://softwaretestingfundamentals.com/white-box-testing/
6. Waterfall Software Development Model. Oxagile. 5.2.2014.
7. Homes, Bernard. Fundamentals of software testing. Weinheim : Wiley-VCH. 2013. (p. 44)
8. Testing Excellence. Waterfall model. https://www.testingexcellence.com/waterfall-model/
9. Spillner, Andreas; Linz, Tilo; Schaefer, Hans. Software Testing Foundations, 4th Edition. Rocky Nook. 2014.
10. Myers, Glenford J.; Sandler, Corey; Badgett, Tom. The Art of Software Testing. 2011. Publisher: John Wiley & Sons, Incorporated (p. 178)
11. QA Lab. Agile testing process. http://www.qalab.co/agile-testing-process.html
12. Montvelisky. Joel. 2016. Why Developers Are Poor Testers and What Can Be Done About It. https://simpleprogrammer.com/developers-poor-testers-can-done/
13. Hoffman, Chris. How-To-Geek. 2017. Beginner Geek: How to Create and Use Virtual Machines (Read 26.03.2019) https://www.howtogeek.com/196060/beginner-geek-how-to-create-and-use-virtual-machines/
14. Asko Soukka. RobotLab Workshop. 2019. https://robots-from-jupyter.github.io/RobotLab-Workshop_2019-01-16.pdf
15. Nicholas Bollweg. Asko Soukka. 2018. Robots from Jupyter. https://github.com/robots-from-jupyter/robotlab/releases
16. https://github.com/jupyterlab/jupyterlab-git
17. https://www.oxagile.com/company/blog/the-waterfall-model/ (Read 21.1.2019)

18. Myers, Glenford J.; Sandler, Corey; Badgett, Tom. The Art of Software Testing. 2011. Publisher: John Wiley & Sons, Incorporated (p. 178)
19. Reichert, Amy. TechBeacon. Software testing for complex, integrated applications? Go modular! (Read 25.03.2019) https://techbeacon.com/app-dev-testing/software-testing-complex-integrated-applications-go-modular
20. Robot Framework. Robot Framework User Guide. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

21. Robot Framework. SeleniumLibrary. http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

22. Robot Framework. BuiltIn. http://robotframework.org/robotframework/latest/libraries/BuiltIn.html

23. Sobolec Nikita. 2019. Engineering quide to writing correct User Stories. (Read 15.3.2019) https://sobolevn.me/2019/02/engineering-guide-to-user-stories

Appendix 1: TESTING INSTRUCTIONS

After a Developer has performed a correction or implemented a new element (let's call it here "element-X") it is recommended to perform some basic testing.

Please, select the appropriate test cases from the list:

OK	Failed	1. Create a small module, which contain at least one element before and after element-X, in addition to Intro and Finish elements.
		2. Run the "Preview" test of all elements in the module.
		3. Test the dynamic editing functionalities, if available.
		4. Test the editing functionalities (copy&paste text in the fields, check if special characters are allowed).
		5. If there is a video(s), test if you can remove /add / play it.
		6. Test if you can scale properly the window.
		7. Test the element tool bar (the red bar), if available
		8. In case of a Log-In element check if you can edit / remove the fields.
		9. Test if you can edit the Title and open the "Link Menu Settings".
		10. Check if you can add / change the background picture.
		11. Check the functionality of every button in the element.
		12. Test the module with the following browsers: IE11, Chrome, Safari, Firefox

Testing responsible:	Task Id:
Project Name:	Testing Date:
Other info:	