Tampere University

Paavo Kemppainen

THE ROLE OF TESTING IN AGILE SOFTWARE DEVELOPMENT

Bachelor's Thesis
Faculty of Information Technology
and Communication Sciences
Examiner: University Teacher Matti Monnonen
May 2020

ABSTRACT

Paavo Kemppainen: The role of testing in Agile software development
Bachelor's Thesis
Tampere University
Software Engineering
May 2020

Software testing is one of the most important phases in software projects because it is used to verify the work that is done and the quality of the work for both the programmer and the client. Testing processes and methods have been advancing side by side software development for years. One of the most common software development types is Agile where the Agile testing also gets its name.

This thesis studies the use of different methods for testing software which together form Agile testing processes. Conclusions in this thesis are based on a couple of well-known pieces on the subject with the help of online material from different opinions. With these source materials, the thesis aims to clarify how different testing methods and processes differ from each other. After this, the thesis represents a consensus on what good and what bad these methods and processes bring to a software project.

The thesis concludes that finding the right testing methods and processes is not easy and one usually has to have some experience selecting them. Every project has its own needs and that way the methods and processes differ between them. One of the most important finds is that the customer is usually in close contact with the testing as they are the ones that define the requirements. A couple of ways to implement this are said in this thesis.

Keywords: Agile testing, software testing, Agile testing advantages, Agile testing disadvantages

TIIVISTELMÄ

Paavo Kemppainen: Ketterän testauksen rooli ohjelmistokehityksessä
Kandidaatintyö
Tampereen yliopisto
Ohjelmistotekniikka
Toukokuu 2020

Ohjelmistojen testaus on yksi jokaisen ohjelmistoprojektin tärkeimmistä vaiheista, koska sillä voidaan varmentaa tehty työ ja sen laatu sekä ohjelmoijalle että asiakkaalle. Testausprosessit ja käytännöt ovat kehittyneet ohjelmistokehityksen rinnalla vuosikausia, joista nykyisin yksi suosituimmista tavoista on toteuttaa ketterää ohjelmistokehitystä, josta ketterä testauskin saa nimensä.

Tässä työssä tutkitaan ketterän testauksen eri testaustapoja, jotka muodostavat yhdessä ketteriä testausprosesseja. Työn pohdinnat ja tulkinnat pohjautuvat muutamaan aiheesta tunnettujen teosten tutkintaan samalla hyödyntäen muiden aiheeseen perehtyneiden tahojen verkkomateriaaleja. Näiden lähdemateriaalien avulla pyritään selvittämään miten eri testaustavat ja testausprosessit eroavat toisistaan. Näiden jälkeen työssä pohditaan niiden etuja ja haittoja ohjelmistokehityksen lopputulokseen.

Työn tuloksena huomattiin, että oikeiden testaustapojen ja testausprosessien löytäminen ei ole helppoa vaan se vaatii useasti kokemusta. Jokaiselle projektille täytyy löytää sille räätälöity prosessi, koska jokainen projektikin on omanlaisensa. Tärkeimpänä löytönä huomattiin, että testausprosessiin täytyy melkein aina sisällyttää asiakkaalla suoritettava testaus, johon työssä on lueteltuna monta eri tapaa.

Avainsanat: Agile testaus, ketterä testaus, ohjelmistotestaus, testaus, Agilen hyödyt, Agilen haitat

CONTENTS

1 Introduction 1
2 Testing practices in Agile development 3
2.1 Agile testing quadrants 3
2.1.1 Quadrant 1 4
2.1.2 Quadrant 2 4
2.1.3 Quadrant 3 5
2.1.4 Quadrant 4 7
2.2 Agile methodologies 8
2.2.1 Test First Development and Test Driven Development 9
2.2.2 Behaviour Driven Development 9
2.2.3 Acceptance Test Driven Development 9
3 Advantages of testing in an Agile environment 11
4 Difficulties of testing in an Agile environment 14
5 Conclusion 16
References 18

LIST OF FIGURES

2.1 Agile testing quadrants [2] 3

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
ATDD	Acceptance Test Driven Development
BDD	Behaviour Driven Development
CIA	Confidentiality, Integrity and Availability
method	a specific approach to methodology
methodology	a system of software development methods
process	part of a specific software development method
software component	can comprise of functions, classes or other parts of a software
TDD	Test Driven Development
TFD	Test First Development

1 INTRODUCTION

Software development is always changing as the general size of a software is always increasing. As the size increases the software is more difficult to handle from many different perspectives. One perspective is the testing of the software. Before the more Agile development methods were invented, people used a process like Waterfall to develop their software. In Waterfall, the testing is a separate part of the development and is done after coding of the main software. After Waterfall, there have been many different processes such as the Spiral model and V-model where testing is done somewhat parallel with the main coding.

After the models such as Waterfall, Spiral, and V-model, came the Agile Manifesto in 2001, whose main point was to be more agile in software development. This process brings with it the ideas such as the customer is satisfied with the continuous delivery of the product at an early stage of the development, the project and its requirements can change during the development. Agile projects don't focus on the documentation of the software but instead, they want to produce functional and high-quality software. [1]

The Agile development model needs a new way of testing the software as in Agile projects the software is usually delivered to the customer in small incrementations before it is completely done. For this reason, the testing is done in parallel with the coding. This places more pressure on the teamwork and communication between the team members and other teams. With so many different opinions based on experience, finding the right methods, processes, and tests can be a challenge, which this work tries to solve.

This thesis analyses different source materials and tries to find how testing is done and how it is supposed to be implemented in an Agile software development environment. The second chapter covers the different practices and methods used in the field of Agile software development. This includes the Agile Testing Quadrants which divides the testing done to four different parts. These quadrants all have their place and different reasons for why they are done. The quadrants are divided between tests that support the team in their development and tests that critique the product. Another way of dividing them is the tests that some are technology-facing and others are business-facing tests. After the quadrants come different methodologies that are used in Agile software development in the following order: Test First Development, Test Driven Development, Behaviour Driven Development, and Acceptance Driven Development. These methodologies are usually quite similar to normal Agile development methodologies as they are usually combined in some ways to the whole software development process. Next is chapter three which

highlights the different advantages of Agile testing that are found either by comparing it to earlier practices used or by discovering new properties that were neglected or not seen previously. After that comes chapter four which naturally after the advantages tries to find flaws, disadvantages, or otherwise difficult parts from Agile testing. The thesis ends with a conclusion of what was found and concluded.

2 TESTING PRACTICES IN AGILE DEVELOPMENT

2.1 Agile testing quadrants

Agile testing can be divided into four main sections that represent different ways and motivations to test in Agile development. The four sections are usually visualised as a diagram with four quadrants as seen in Figure 2.1. Each of these quadrants has a role in the Agile testing process. [2]

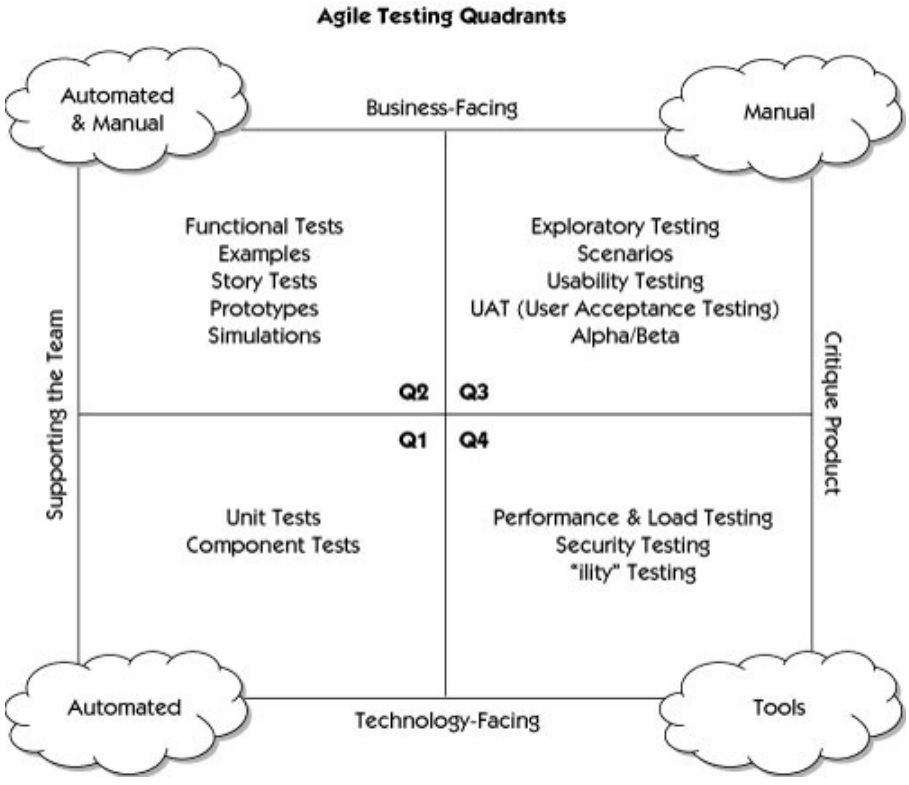

Figure 2.1. *Agile testing quadrants [2]*

Figure 2.1 shows how the vertical axis divides the quadrants to technological testing and business side testing. The technological side includes quadrant Q1 which includes automated testing that is useful for the developers themselves and Q4 which tests the overall performance and security of the system. The business side of the quadrants includes Q2 which tries to test that all the requirements and functions work as the customer wanted and Q3 which involves the customer in the testing process such as a beta tester or general acceptance tester before accepting the project.

2.1.1 Quadrant 1

Quadrant 1 includes technology tests that support the development team to build the foundation of their software design and development. These tests include Unit and Component tests which with automation and continuous integration help the team in delivering quality software to the customer on time. [2]

This quadrant helps the developer team to make sure their code is working before they publish it to the customers. Unit and component testing are usually done in the same programming language as the main software itself. For this reason, someone with no knowledge of this programming language can't understand these tests as they are not meant for them to understand. As all the upcoming Agile testing practices this testing is done during the developing process. [2]

In terms of roles, Unit testing verifies the functionality of a small portion of the software. These functionalities can be as small as testing a method of a class or even a part of the said method. [2] They are done so that other system parts don't interfere with the part in testing. These parts can be classes and their methods in object-oriented programming and functions, modules, and scripts in non-object-oriented programming. [3]

Component testing focuses on the general design of a specific part of the software and the behaviour between the classes or their methods. [2] These tests are done to determine the usability of the component from a standpoint of a non-developer such as the person writing the test might not know about the internal architecture of the component as in black box testing. Component testing is usually done by the specific test teams or the tester specialist in the development team. These tests are performed after the unit tests. [4]

2.1.2 Quadrant 2

Quadrant 2 is one-layer higher testing compared to Quadrant 1 but it is also for the development team. This type of testing includes a lot of business and customer side of testing which aims to improve the quality of the software to a level the customer has required it to be. Tests in this quadrant for example include functional tests, story tests, and prototypes. These are meant to be understandable by people who don't know how to code in the language that the software is written. This helps to deal with business experts so that even they can help to do the tests together with the main coders. [2]

Functional tests are for testing the functionality of the software. Functionality testing is done as a black box testing that tester does not know the underlying architecture of the program. [2] This seems similar to component testing in Q1, but functional testing tests the software on a larger scale, and it tests the whole functionality required to meet the user requirements for a given user story for example [5].

Story tests can be done by the customers and they usually depict the required feature and

how they would like it to work. The work done in user story writing is crucial to determine the customer needs for the software as if this part is not handled with care, the software could face multiple changes to its core components. These stories give the programmers the needed guideline as to what they need to code and test and conditions as to when they are done with a feature and can move on to the next feature. [2]

Prototype testing focuses on the user interface and most importantly the user experience when they navigate the user interface and try to carry out the user story requirements [2]. This testing can have multiple phases. When the developer team starts to think about the user interface the prototype can be as simple as a paper presentation that has multiple paper slides that try to mimic the initial user interface and its functions. This way is easier and cheaper to test out multiple different user interfaces without any coding at all. As the project moves forward the prototyping advances also as the developers have a better idea of the end product. The prototype can advance to the mock-up stage where it is done with digital tools and has colours but has no functionality. After that, the functionalities come to the prototype with a click dummy stage which shows how the user can move around in the software features. This stage can be the first stage where the prototype is shown to the customers to get their feedback. User experience testing should be done by the user experience specialists or crowd testing can be utilised at this point. In this stage, the crowd represents the end-users. This is a good way to determine if the user interface matches the end-users' expectations. [6]

As this quadrant and quadrant 1 are meant to be for the development team this quadrant also is to get fast feedback and enable easy troubleshooting for the team. For this reason, quadrant 2 tests should also be automated. The tests should be run frequently as a continuous integration so that the possible problems could be fixed as soon as possible. As the software starts to grow out and not all the people working with it can maintain a full understanding of the current software these tests help avoid the possibility that some change in another part of the program causes bugs or other problems that in the end showing to the end customer. [2]

2.1.3 Quadrant 3

Quadrant 3 focuses on the business side of testing as its main point to make sure the product that is being delivered to the customer meets the customers' needs and requirements. The testing done in Quadrant 3 can identify mistakes, misunderstandings, or missing functionalities from the software. This can be tested by the developers themselves but can also be done by the customer and the end-users. This way the product can be emulated in a real-world situation to affirm that the product is and works like it was supposed to be. This Quadrant includes tests such as Exploratory, Scenario, Usability, User Acceptance and Alpha or Beta testing. [2]

The Exploratory testing focuses on the testers' ability to come up with different test cases during the test procedure. This often requires tremendous effort and ability from the

tester to achieve the wanted result. Exploratory testing process is not random as the tester must have some goals in mind when they start to explore different routes in the software. As this process is the most complicated part of testing it also usually reveals the most serious bugs. The process usually starts at the beginning of the development cycle where testers get their hands on a small piece of testable code and after testing that part, they might get new ideas on what parts to explore next. This seems similar to the way an end-user might test the end software and that way is good at finding bugs. [2]

With the help of business-side users, the development team can create scenarios and different workflows that try to represent the possible real-life situations as well as possible. These scenarios are then tested with the most realistic data and flow of the program. It is no use to test scenarios with data that is not realistic as it might not reveal bugs or misunderstandings that are hidden. When the developed software gets bigger with different features the flows usually expand as well and this means that different decisions during the flows behave differently. These data flows and process flows can be visualised with diagrams to help analyse them better. [2]

Usability testing in Quadrant 3 includes testing that critiques the software instead of helping the initial programming of the user interface and other areas that affect usability such as in Quadrant 2. As the competition can sometimes be harsh, especially in commercial software, usability can be a determining factor whether a user picks a certain software instead of another one. If there are any competing or otherwise similar products on the market, it might be useful to take a look at their usability and compare it to the software that is being developed to get an idea of how their usability differs. [2]

A proven method to do usability testing is to utilise persona testing. Persona testing aims to describe a person from every demographic group that would be using the software on a somewhat regular basis. These groups have different needs and different experience levels that might affect their experience in usability when operating the software. The ease and smoothness of navigation in the software is also a big part of usability. This includes links and buttons going to the places they are supposed to take the user. As with other usability tests the end-user can be used in this stage to test out the navigation while someone is taking notes of how the user ended up doing a certain task on the software. [2]

User Acceptance Testing has close relations to Acceptance Driven Development and is usually considered a part of it. User Acceptance Testing focuses on the later parts of software development cycles as it has actual end-users testing the software. This stage of testing assesses whether the software can handle the daily situations, user scenarios, and is sufficient to be deployed to the customer. The end-users, that are doing the testing here, would preferably be the actual end-users as in people who have to use the software in their daily work. This can include managers and more higher-ups depending on the software, but they should also be from the actual user groups that commonly are lower on the hierarchy. [7]

Alpha testing takes place quite late in the software development. In this stage, the main features should be implemented, and the testing should focus on them. The testers could comprise of internal company members, but they are not usually part of the actual development team. The main point of this testing is quite similar to User Acceptance Testing that it should focus on ensuring that the software behaves as intended when it is used by a wider range of users. Major defects found in this stage are to be fixed before the software can continue to Beta testing. In Beta testing the software is exposed to even wider user groups with the hope that final bugs and defects are found in this stage to get the software ready to be released to the public. [8] The main difference with User Acceptance Testing and Alpha/Beta testing seems to be that User Acceptance Testing term is used when the software is done in close collaboration with the customer and Alpha/Beta term is used when the software is at the end released to the public market and such can have unsuspected user groups.

This testing is the hardest part to automate as the testing is done by people and a big part of the testing is to test how the program feels to use. Still, some tools help with this part of testing such as predetermined setup test data and such. While acquiring the testers for this part one must think of all the user groups that will be using the end product and as such this becomes a whole new area of expertise to analyse. The users must be analysed during their testing to see if they use the software differently than the developers thought they might and in the end they might have to interview them to get some solid feedback and genuine reactions. As in Agile, the testing is done during the development period this user acceptance testing is a good way to get new possible ideas and improvement suggestions before the whole product is ready. [2]

2.1.4 Quadrant 4

Quadrant 4 aims to test the technology of the software in areas such as performance, security, and robustness. This part of testing is on the shoulder of the developers as usually, the customer takes them for granted when they talk about their requirements for the software. For example, they don't usually talk about how high a response time a website should have, they just expect it to be good. For this reason, it is important to clarify with the customer on each of the areas covered in Quadrant 4 how high they are on the list of importance so that the developers can focus on these instead of guessing what the customer thinks as important. [2]

This quadrant covers a lot of different non-functional testing and every one of them might not even happen for every software project. The ones that are tested depends on the customer and the type of software being developed. [2] For example if the customer doesn't value some non-functional requirements high that part might not have any resources allocated to its testing. Some software just doesn't need some types of non-functional testing such as offline software doesn't need to worry about response times over the network.

Performance testing helps identify the possible bottlenecks in the software such as algo-

rithm run times or overall software response times. What performance is required usually depends completely on the customer's wishes. Load testing aims to test the whole system's behaviour as the system gets more and more users at the same moment. A more extreme load testing called stress-testing tries to cause more stress to the system than was originally planned and required to test its scalability. These tests should be done during the development process, if possible, to get early feedback on the performance side of the software's architecture and design decisions. If these problems are not solved early on the development cycle it keeps getting more expensive to solve them eventually. [2]

Security can be one of the hardest testing stages to get right as the software is still most of the time done by humans and humans are prone to make erroneous decisions or bugs. With most of the data being on a non-physical form such as in cloud databases or just on local databases the importance of that data being securely handled and transmitted such that all the security parameters are fulfilled. [2] These parameters include confidentiality which means that the data is not available to anyone else than those who have permission to it, integrity which means that the data is not altered by an unauthorised party which includes modifying or deleting it, availability which means that the data is available to the authorised parties when they so desire. These three parameters form commonly known Confidentiality, Integrity, and Availability (CIA) triad which has many other extensions that define more important parameters. [9] With these different parameters the testing has some requirements it must achieve. The testing itself is usually so demanding the tester's skills and experience that small development teams don't have the skills themselves and they have to rely on security specialists to do the job. In bigger organisations, there is usually a team of security experts that either test the software or help out the development teams in testing the software's security. [2]

To make the software more robust it may need to be tested on its maintainability, interoperability, compatibility, reliability, and installability which all should have their requirements from the customer. Whichever of these parts is being tested it is usually suggested to do these incrementally as in as the project evolves during the development these tests are being thought through and maybe even implemented. [2]

As in Agile development, the most important thing is to listen to the customer and extract the user stories and requirements and focus on them because that's what they pay for, it can be sometimes forgotten that the customers expect somethings to a certain way. Because of these reasons the developers have to have a stand on how to make sure these 'hidden' requirements are fulfilled. [2]

2.2 Agile methodologies

As Agile testing goes hand in hand with Agile software development it can sometimes be hard to differentiate Agile testing methods from Agile software development methods. The next upcoming practices are just some of the most common bases for Agile development

and at the same time for Agile testing. [10]

2.2.1 Test First Development and Test Driven Development

Test First Development (TFD) focuses on test preparation as in this practice the tests are done before the actual coding takes place. The tests are made when a story is developed by team members skillful in analysis, designing, building, and testing. Sometimes the customer can be involved in the test creation process to give them some assure them that their criteria are met. [11]

After the tests are done, the test-build cycle begins in which the software is implemented as the tests direct it to be. If the tests fail during the development the software must be fixed to pass the tests. After the implementation works with the tests, all the bugs have been fixed and the customer is satisfied with the result of the story, the story is done. [11]

Test Driven Development (TDD) has the same priorities as TFD. TDD's main focus is the automated unit and component testing in testing quadrant 1. In this practice, the tests are also done before the coding but TDD also has a step of refactoring the code as the design evolves during the coding. The automated tests provide better guidelines for the development team to provide quality products. [11]

2.2.2 Behaviour Driven Development

Behaviour Driven Development (BDD) makes sure that the software delivers the most accurate representation of the customer's requirements as possible. This goal is achieved by utilizing people from many different groups in the development process. These groups contain people with analytical, design, coding, and testing skills. Behaviour Driven Development aims to construct a bridge between these groups and that way they can work together and achieve the best possible outcome. [11]

The testing itself focuses on scenario testing which was covered in testing quadrants. This testing type aims to make sure the software behaves as it was designed to behave. [11]

2.2.3 Acceptance Test Driven Development

Acceptance Test Driven Development (ATDD) focuses on the collaboration with the customers and testers. They all work together before the programming starts and create cases of how the customer might use the program and how they want it to behave. This a great way to get the most detailed description of how the software should work in certain scenarios. [12]

User acceptance tests can be used in different areas of the testing quadrants. For example, they can be applied to unit, integration, or user interface testing levels. They can also

be used to validate proper inputs and outputs in the customer's environment. The main point is to affirm that the customer accepts the software.

3 ADVANTAGES OF TESTING IN AN AGILE ENVIRONMENT

Agile testing has many similar advantages as Agile software development has. With the development done during the development cycles, the software must be tested and accepted during the same period. This requires good communication between the testers and the development members. Close low-level communication may reduce the need for profound and laborious documentation for the whole software and testing plans saving time. [13] As Agile development usually works in short development cycles; any amount of saved time can be used more efficiently on the things that matter such as more profound exploratory testing or overall better user experience.

Agile testing is very flexible with many different methods and procedures such as ATTD, TDD, TFD giving the possibility of choosing the best testing practice for the project and specifically for the team in question. With experienced teams, the Agile methods can be very time and money efficient as they are not bound by unnecessary routines and protocols. [13] For example in case 8.2 of Linz Tilo's book a team relatively new to Agile methods was able to find suitable methods for their needs after a year of experience with Scrum. After they got used to the chosen methods their overall quality of the software improved. Even after they found the best methods for them at the time they were not ready to stop improving their processes and had already new plans for improving their software used for system requirement describing, Continuous Integration process and overall improvements to test environment. [3] This sort of lack of boundaries and breaking the norms of the traditional software development methods, such as Waterfall or even other Agile methods, can bring forth new and experimental approaches to new and old problems. As these new approaches become more widely used as they get passed around in the software development circles for some time can then become the new norm that can become the new Agile method that everyone tries to copy. With this, even the worldwide community of software development can improve when they are not stuck on following the old and proven methods which might work in some cases but not in others.

With not so structured workflow and the need for fast product output such as in Agile development, there can be a temptation to take shortcuts or implement fast bad solutions to the problems in the code. If this is not prevented it can lead to massive amounts of technical debt on the project. With the four quadrants in mind and following their guidelines in testing the software, one can reduce the technical debt build-up and other unforeseen

problems efficiently. The technology tests can help keep the code maintainable while automated unit tests help reduce technical debt. With load and stress testing, the team gets early information on how the software handles the pressure and can help steer away from nonapplicable architecture solutions. [2]

At times developers can become lazy, are short on time, or face other obstacles, and think about testing later which could lead to testing having less time than optimal. In Test Driven Development and most other Agile development methodologies, the tests are done beforehand which tries to negate this problem. With tests that are done correctly before coding, the code itself usually has better quality as they have to pass the tests before they can continue to the next feature. [14]

By following a strict Test First methodology, when doing unit testing, the developers can increase their testing effectiveness. Automated unit testing with proper inputs and expected output values improves the testing as they don't have to keep doing manual testing which takes a lot of time when done multiple times. These automated tests also use different variations of inputs and invalid inputs making them more robust. [3]

By actually writing the tests and not just testing them by hand the developer may be able to see if something is missing from the code even without running the tests. When the test cases are in use before the coding starts the developers have some kind of progress feedback. As they write their code and test it, the code may pass a test which assures them their code is probably doing something right. [3]

When the Unit tests have a wide enough coverage, they can be seen almost as a specification for the code. With specific input, the test expects a specific response from the software. This means that the tests specify the software's expected behaviour. With this, the developers can read the tests and they get the specification of the chosen piece of software. [3]

The tests and specifications being the same, it is easy to see the Application Programming Interface (API) of the piece of software, usually a class or something similar. By writing the tests before coding, the public methods that are used will have their expected inputs and expected behaviours documented in the tests. [3] This can help with coding software parts that interact with the said piece of software as they can read the tests and write their API calls according to this specification.

With Agile development, the testing has been moved to collaborate with the main development itself. This creates the need to integrate the testers to the developer teams. It is not unusual to see teams consisting of a tester and a few developers working together. [15] As the team works together, they all share the responsibility of the code's quality. With new responsibilities team members that didn't before think about certain parts of the software were aware of the problems, their solutions might cause to the overall product. This also allows the team to learn new skills as it now has people with different roles and experiences as learned in Case 8.1 in Linz Tilo's book. [3]

With the skill-sharing in these new collaborated teams or with external coaching, the test

cases have improved. This has led to less time being used for testing as seen in Case 8.5 in Linz Tilo's book. [3] This then correlates to faster and better testing and that way fewer expenses on testing.

The combined team's developers usually work so that as they write the code, they do some kind of unit testing and after that, they pass the code to the tester. The tester then proceeds to test the code further with other test cases [2]. As the software develops further, the small pieces of code are tested against multiple tests many times which provides better quality code overall with the presumption that the tests are viable.

While working in teams the communication between the testers and developers is better than if the tester was in a separate team and as soon as the tester finds a bug, they can just inform the person responsible for that piece of code to fix it. This fast feedback is the backbone of Agile testing and it usually provides fast and easy defect fixes. [2] Early testing also lessens the chances of cascading bugs where a bug is detected late after which it is fixed but then this bug or the fix itself causes some other unwanted behaviour as other parts of the software might have been using the buggy code. Sometimes reproducing the found bug can be difficult so with continuous testing there usually won't be that much new code to check to find the part which causes the bug. [15]

4 DIFFICULTIES OF TESTING IN AN AGILE ENVIRONMENT

The disadvantages of Agile testing has mostly sprung from the nature of Agile software development itself as they are so tied together when using agile methods. The biggest disadvantage of Agile testing is the need for an experienced team to succeed in it. [16] This stems from the fact that Agile testing practices are not set in stone on how they are implemented in real work situations. The most suitable practice may take a lot of time and sometimes the good ones are found by trial and error. This can cause a lot of lost effort and money as the product is delayed. There are cases when companies have employed coaches that know Agile methods to the inexperienced teams to help solve this problem as in Cases 8.5 and 8.6 [3]. This creates more expenses at the start but could save a lot of time and money in the long run without the trial and error part.

With inexperienced teams comes the need for even better communication between the teams and their members. Inefficient communication between the team members most often causes them to misunderstand their and their clients' needs which can cause incorrect testing. [17] Even if the team is aware of the requirements at the beginning of the development, it is like Agile development that requirements can change. [16] These changes can cause the previously done testing to obsolete as the feature could change dramatically or be removed.

Agile testing requires more effort on an individual level than the traditional ways of testing as the testers and developers have to have a deep collaboration between them during the whole development process. When a bug is found by the testers it needs to be communicated to the developers and depending on the severity of the bug the whole project could be on hold until it is fixed. Usually, proper testing from the start prevents major bugs but it can still happen. [16] It is common for the end-user to test the product during its development. The resources needed to do User Acceptance tests are quite high compared to other types of testing in the development house if done thoroughly. The costs comprise of researching the proper user sample from end-users, setting the test environment, letting the customer test the product, analyse user behaviour during the test, conduct some kind of interviews after the test and finally analyse and come to some kind of a conclusion about how the software fared in the real test environment. [2] For example in Case 8.5 the company had a separate team that was in charge of system testing [3].

The lack of documentation in Agile development transfers to Agile testing as well and can sometimes cause issues in some cases. When the testing is done during the development process there rarely is any time to document what, how, and why something was tested. This problem comes in question when a new team member or team is introduced to the project. When they begin to grasp the overall picture of the testing done to the software it can be very difficult to maintain a clear picture of it. [16]

5 CONCLUSION

The focus of this work was on finding out what different practices and methods Agile testing has in store for the software developers and testers. The main subjects that were under research were the Agile testing quadrants, Test First, Test Driven, Behaviour Driven, and Acceptance Driven Development. With these testing and development methods in mind, the work aimed to get a clearer picture as to how Agile testing works and how it might be beneficial or in some cases disadvantageous for software development projects. This was done by comparing literature from the research field and other software development focused sources.

Agile testing proves to be most beneficial to the project when done correctly. Correctly chosen methods and practices with the chosen Agile development process itself offer multiple advantages to other testing practices. Advantages appear as better software quality as it has fewer defects, better user experience, maintainability, performance. These advantages are mostly due to the idea of testing while the development process is ongoing and that the Agile development processes usually involve the customer more than other processes which makes the requirements flexible and clearer for everyone. It is also possible to attain faster delivery of products, less expensive results, and overall more efficient development processes with Agile testing. Faster delivery of a product is not an instant change when coming from non-Agile processes, but it comes with experience as the development teams get familiar with their work which might take more than expected. As the teams get experience and the work flexible the team gets more efficient. The lack of everything non-essential, such as some documentation, to the development and testing process the developers can focus on the essential things making the efficiency higher. With all these factors in place, the overall product is going to be cheaper than normal.

There are disadvantages to Agile testing as there for almost anything in the world. These disadvantages don't necessarily come from the methods or practices themselves but the people implementing them. As said in the previous chapter Agile development is quite flexible but with flexibility comes the need for experience so that the development teams know how to choose the right workflow that suits them. The Agile Development altogether require more individual effort and places more responsibilities on the testers and developers. With this, the need for good communication and work atmosphere is detrimental to the development and testing processes. There might be a problem of lack of documentation with Agile testing as when new members join the project, they must be

somehow introduced to it.

Overall the covered materials gave a good overall view of the field and cover them in more detail than needed for this work and as such can be used in further research in the future. Agile testing proves to be a difficult area to cover accurately as the field is always evolving and as such the terms might differ from source to source. It is not uncommon for two sources to have the same base development process but they still have their interpretation of the process. This might cause difficulties when researching this field further. The work itself was able to cover the most common methods and processes used in Agile testing and gather some thoughts on how they could be used and what benefits and disadvantages they offer.

REFERENCES

[1] *The Agile Manifesto*. 2001. URL: https://agilemanifesto.org/ (visited on 09/02/2020).

[2] Crispin, L. *Agile testing : a practical guide for testers and agile teams*. eng. Addison-Wesley signature series. Upper Saddle River, NJ ; Addison-Wesley, c2009. ISBN: 1-282-30313-9.

[3] Linz, T. *Testing in Scrum : a guide for software quality assurance in the agile world*. eng. First edition. Santa Barbara, California: Rocky Nook. ISBN: 9781492001546.

[4] *Difference between Component and Unit Testing*. URL: https://www.geeksforgeeks.org/difference-between-component-and-unit-testing/.

[5] *Functional Testing*. URL: http://softwaretestingfundamentals.com/functional-testing/ (visited on 05/02/2020).

[6] *Prototype testing – What, how and why?* URL: https://www.testbirds.com/blog/prototype-testing-what-how-and-why/ (visited on 05/02/2020).

[7] *User Acceptance Testing – How To Do It Right!* URL: https://usersnap.com/blog/user-acceptance-testing-right/ (visited on 05/03/2020).

[8] *The Difference Between UAT, Alpha Testing, and Beta Testing*. URL: https://betsol.com/difference-between-uat-alpha-testing-and-beta-testing/ (visited on 05/03/2020).

[9] Rhodes-Ousley, M. *Information Security*. eng. 2nd ed. Place of publication not identified: McGraw Hill Osborne. ISBN: 0-07-178435-7.

[10] URL: https://www.standandstretch.com/agile/advantages-and-disadvantages-of-agile-testing/.

[11] Measy, P. *Agile foundations: principles, practices and framework*. eng. 2015. ISBN: 1-78017-256-7.

[12] Pugh, K. *Lean-agile acceptance test driven development : better software through collaboration*. eng. Net objectives lean-agile series Lean-agile acceptance test-driven development. Place of publication not identified: Addison Wesley Professional. ISBN: 1-282-94768-0.

[13] *Agile Testing – Principles, methods advantages*. URL: https://reqtest.com/testing-blog/agile-testing-principles-methods-advantages/ (visited on 04/02/2020).

[14] *Test First Development*. URL: http://tutorials.jenkov.com/java-unit-testing/test-first-development.html (visited on 04/02/2020).

[15] *Advantages of Agile Testing*. URL: https://www.optimusinfo.com/blog/advantages-agile-testing (visited on 04/02/2020).

[16] *Advantages of Agile Testing*. URL: https://www.standandstretch.com/agile/advantages-and-disadvantages-of-agile-testing/ (visited on 04/02/2020).

[17] *A Comprehensive Guide to Agile Testing*. URL: https://www.sam-solutions.com/blog/guide-to-agile-testing/ (visited on 04/02/2020).