Creating a new web system and developing it further – Tester's perspective in a diary form

Alpo Remes

Haaga-Helia University of Applied Sciences
Haaga-Helia Bachelor's Degree
BITE
2024

Abstract

Author(s)
Remes Alpo

Degree
Bachelor of Business Information Technology

Thesis Title
Creating a new web system and developing it further – Tester's perspective in a diary form

Number of pages and appendix pages
54 + 1

Thesis describes the author's work experience in a state-owned institution from Finland. Institution is the Finland's Ministry of Interior and is referred to as "institution" throughout the text. The institution was in the process of acquiring a new software system from the vendor, Sofigate. The system would be large and complicated, requiring a lengthy development process. The institution required additional assistance in the project and hired the author as an intern. After the internship was over, author got hired as a designer, continuing in assisting in the project. The timeline of the diary was mostly during when the author was working with the designer title. First time window covered was between 17.10.2022-22.11.2022. After, a time skip was made, and the next window covered was between 08.03.2023-24.03.2023. The skip allowed the coverage of two user acceptance periods.

From each week, an analysis was made based on the tasks reported during the week. Observing the different weekly analyses as a continuation, improvements could be noticed. These observations, including improvements between the two user acceptance periods are discussed.

Objective of the thesis was to first examine different testing techniques with theoretical framework, mainly from the perspective of manual testing. With the assistance of theoretical framework, diary entries were reported. Working as a tester from the customer's side, a more unique perspective could be shed to testing. Even if using agile methodologies would generally increase customer involvement compared to more traditional methods, customer performing testing throughout the full development lifecycle instead of just user acceptance period was more uncommon technique.

Keywords
software testing, user acceptance testing, defect, agile, software development project

Contents

Terminology

Testing – Purpose of testing is to ensure adequate level of functional and non-functional software behaviour (Lonetti & Marchetti 2018, 1).

Defect - An imperfection or deficiency in a work product where it does not meet its requirements or specifications (ISTQB 2022).

Software development lifecycle - The activities performed at each stage in software development, and how they relate to one another logically and chronologically (ISTQB 2022).

Test case - A set of preconditions, inputs, actions (where applicable), expected results and post-conditions, developed based on test conditions (ISTQB 2022).

Manual and automation testing - Manual testing is done manually by humans, automation testing is doing similar testing but using tools that automate the process.

Acceptance testing – Final stage of testing before deployment of a product. The goal is to find out if the software meets the requirements of the customer, which is why customers are the ones doing it mostly (Lonetti & Marchetti 2018, 5).

GUI-based testing – GUI (graphical user interface)-based testing is like traditional software testing, but there are some differences related to defining appropriate test criteria (Liebel, Alégroth & Feldt (2013, 17). Still, testing through GUI is done by testing beta versions of the product (Liebel & al. 2013, 18). This is the same as manual testing.

Agile software development – A group of software development methodologies based on iterative incremental development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams (ISTQB 2022).

Sprint – Short cycle in Agile, where a set of functionalities are developed and then tested during that cycle, limiting the difference between the needs of the customer and the developed end result (Edeki 2015, 22).

1 Introduction

Diary discusses my work as a tester, being one part of a large software project between the vendor and customer. Work was done on the customer's side, shining a light to a less discussed way of testing – not every project has a dedicated testing team from customer's side helping developers for the whole development lifecycle. Generally, the entity developing the product handles the testing almost completely as well. Only in acceptance testing is the customer the main actor. In other periods, testing is mainly handled by the vendor (Naik & Tripathy 2008, 26). While in our project vendor performed testing as well, new functionality of each sprint was also tested by the customer. Naturally, our testing didn't cover every type of testing, as we didn't have access to everything required. Thus, we conducted mainly testing related to black box testing, meaning testing the interface and its inputs and outputs. In agile methodology, it's common for the customer to see the process of software developing by each sprint (Edeki 2015, 22). In our project, customer involvement went further than demonstrations though, as we had a dedicated testing team for the software from customers side, the one I was part of.

Diary timeline is as follows: 17.10.2022-22.11.2022 for the first 5 weeks. After this, a break was taken, and the diary was continued on 08.03.2023-24.03.2023. Both timeframes ended at an acceptance testing period.

1.1 Institution and team details

My institution was in public sector, with around 260 employees. There was an IT department, but the main actors in the project were not in IT, besides me. However, some had previous knowledge of software implementation, as they had been doing similar projects to this before. For me, it was the first time doing something like this. My expertise was in IT though, so I was able to offer unique capabilities to the customer's team. I had studied and performed testing in Haaga-Helia, and that experience was essential at my work. My programming courses taken were useful as well, as they gave me an idea of what is possible, and what would be easiest to do for the programmer. I worked in the institution for total of 11 months, first starting as an intern, and later with a title "designer". We had a smaller team dedicated to the project, with 3 and later 2 people plus our supervisor. In addition, people joined at different times throughout the development lifecycle with various levels of dedication depending on the need. For example, in meetings defining requirements, people responsible for the area requirements applied to were closely aboard. In acceptance testing, a significantly larger number of employees were working on the project, testing areas that every respectable employee was an expert in. The system was built on the base of ServiceNow, with modifications based on our needs.

Our smaller group was responsible for the project. In the group, the responsibility was on the other members besides me, and the bigger picture was on the supervisor. We worked closely together, and I could always ask them if I was unclear of something or needed assistance. Work was mostly remote, as this was during covid, but we had a mandatory workday in the office once every two weeks.

Our group worked very closely with the vendor. Our project manager held meetings with the project manager of vendor many times per week. During the requirements stage, we had meetings almost every day and some days several meetings discussing various parts of the project, including requirements of the system. During development we had sprint meetings, discussing the functionality to be added during that sprint. We also had meetings if an added functionality was unclear, and it wasn't solvable with emails or equivalent. New test cases were added to the same system that defects were reported to. This way, everything was documented and easy to find and keep track of. The same system had old test cases saved, giving the ability to check them even after accepting and closing them. More than once, I went back to an already accepted test case, if I needed to understand the intended functionality or even to reopen the test case. For project managers, having every test case in the dedicated system helped to keep track of progress and to give clarity on the scope of each sprint. With test cases being about requirements cut into small tasks gives the possibility to determine the percentage of the software completed at any point of the project (Edeki 2015, 23).

1.2 Responsibilities and project objective

My responsibilities included black box testing and inputting defects to the vendor. During acceptance testing periods and during system testing, when people outside our smaller group performed testing, I gave assistance when needed. Assistance included teaching about the new software environment, clarifying orders and assisting in reporting defects. On the second acceptance testing period, with help I also planned and managed the execution of the phase. Going into my internship and then full-time employment, my experience was from courses in programming and design. After the employment ended, I did a course on testing and researched the topic more. Based on my new knowledge, I was able to give this thesis a more academic background and foundation, but while I was working, I didn't yet have that additional knowledge. While working, I based my actions on how I had done my own projects and how testing had been done in the few courses that included testing.

The main objective of the project was to deliver a finished project, a software system that would be put immediately to use after launch. There was a strict timeline for finishing that could not be

surpassed in any circumstances. System was to replace the old one that was used for the same purpose, submitting official applications for receiving significant amounts of money.

1.3 Key concepts

There is variety in defining testing. According to Naik & Tripathy (2008, 7), testing has two important roles in achieving and assessing quality of a software project. First, testing is used to improve the product with faults found from testing get fixed. Second, testing is used to assess the quality of the product and how good it is. Jamil, Arif, Abubakar & Ahmad (2016, 177) defines testing as a process to evaluate if system meets its specified requirements or not. Lonetti & Marchetti (2018, 16) describe testing as evaluating the quality of product and increasing confidence in the functionality of the software. Additionally, Lonetti & Marchetti (2018, 16) remind that testing cannot show the absence of defects, but rather that defects are present. In other words, software cannot be fully cleared of bugs and issues, but testing can put the software into an acceptable state from the perspective of functioning.

Software development lifecycle is defined as “The activities performed at each stage in software development, and how they relate to one another logically and chronologically” (ISTQB 2022) in the ISTQB glossary. Lonetti & Marchetti (2018, 16) explain that testing is done differently at each stage of the lifecycle. These different techniques serve a different purpose, and they are all important to ensure the quality of the product. According to Lonetti & Marchetti (2018, 16), these different testing levels are all equal, and results from one level can’t be supplied to another directly, as each level addresses different types of failures.

Manual testing means testing by humans. Automated testing is becoming more popular, but it can’t fully replace manual testing. According to Naik & Tripathy (2008, 25), some test categories are not suited for automation at all, like usability or compatibility. In fact, only 50% of the system-level test cases can be automated. Automated testing can be useful as well. Automation is very suitable for repetitive type of testing. It also increases coverage for testing and has other benefits, like increased productivity of testers and reduced cost of maintenance (Naik & Tripathy 2008, 24).

2 Description of the initial situation

In the first section of this chapter, the current situation at work is described and my main responsibilities are discussed. Then, main testing techniques related to my work are explained, supported by various academic sources. Finally, stakeholders related to my work are identified, with common interaction situations I share working with them.

2.1 Analysis of the current work

Relevant tasks to this diary in my work relate to testing. In the beginning, manual testing is performed, taking advantage of various techniques in the area. These include but are not limited to exploratory testing, error guessing and regression testing. Mentioned techniques are explained in the next section. Later as more experience is gained, tasks diversify further – test planning is made, alongside with creating instructions for testing. Managing a user acceptance period is executed. As part of the responsibilities shift towards management, help is also provided to other testers in small and larger meetings. Additionally, meetings related to testing are hosted. These mainly include meetings during user acceptance periods.

To overcome these tasks, following know-how has been acquired prior to covered period:

- Development courses. During development courses, from own programmed software, experience is obtained in understanding how a system is built and what are its limitations.
- Design courses. During design courses, testing was conducted for different prototypes. In the area, common procedures and best practices were covered.
- Prior work experience. Before the covered period, experience has already been gained in working on the project. With this experience, repetitive issues can be identified and solved. Working in the same project for an extended period has increased understanding of behaviours for individuals involved in the project.

To reach the next step in becoming a test professional, test automation should be learned. At times, work with repetitive nature is covered in the diary. Such work would be beneficial to automate. The employment is on the customer side, thus not having access to the code and back-end of the system in general. Thus, automation would be difficult to achieve by our group. However, the benefits of automation are still apparent in diary for parts of testing. That’s why automation should be learned to perform a more efficient and coherent testing in future projects, in other places of employment. For this project repetitive work is done manually, because it wasn’t feasible to use automation for us.

Another requirement for becoming a successful test engineer is mastering test planning. For the project covered in the diary, test cases are created by vendor, by the developers. A good idea is gained of the nature and content of the test cases, but a deeper understanding should be learned for future projects.

Responsibilities on the project aren't as extensive as responsibilities for most test engineers, with automation testing, among other things, not included in the job description. Thus, in the frames of the job description for this specific employment, my competencies are in the level of skilled performer. My understanding of the work expectations is in a high level, and the skills I possess enable reaching those goals.

2.2 Testing discussion

Manual testing is still a widely used technique, even though automatic testing is making a rise. There is quite many publications about automatic testing, but not so much about manual testing. Automatic testing can be cheaper in many cases, and easier from a certain point of view, which is why there's more interest for it than for manual testing. From the case study done by Liebel & al. (2013, 21) it is discovered, that automated GUI-based system testing is done mostly on small scale if at all. Case study was about GUI-based testing, and manual testing is essentially always GUI-based, (especially acceptance testing) which is why the findings of the study can be applied to manual testing in general. Liebel & al. (2013, 21) find, that manual GUI-based system testing is still widely used technique in system testing. Large amount of use cases was executed manually in the companies interviewed by Liebel & al. (2013, 21). Naik & Tripathy (2018, 395) state that test cases should only be automated if there's a clear benefit in doing so, as some cases are easy to automate, and some can be difficult. Given that manual testing is currently receiving less coverage compared to automation, emphasizing manual testing in this thesis can provide valuable insights.

Testing on high level can be divided into two groups: static and dynamic testing (Lonetti & Marchetti 2018, 3). Static testing refers to examination of documentation and code, not actually running code. This kind of testing is quite tedious and error prone, so the more it can be automated, the better (Lonetti & Marchetti 2018, 4). Dynamic testing refers to executing the program or part of it to obtain information about it (Lonetti & Marchetti 2018, 4). In dynamic testing, unfound issues inside the documentation or code will be revealed in the form of faulty or unwanted processes.

2.2.1 4 levels of testing during software development lifecycle

As demonstrated in Figure 1, there are 4 levels of testing during software development lifecycle: unit, integration, system, and acceptance testing (Naik & Tripathy 2008, 16). Lowest level of testing is unit testing. It's done on singular components on the system (Liebel & al. 2013, 17). Unit testing

has two phases, complementing each other: Static unit testing and dynamic unit testing (Naik & Tripathy 2008, 52). Integration testing means combining different units into a larger entity. Ideally unit tests have already revealed enough defects, that integration tests should discover faults in the implementation between units, not the units themselves (Gerardo Orellana, Gulsher Laghari, Alessandro Murgia & Serge Demeyer 2017, 1). Some issues in a unit can't be found with unit testing alone due the limitations of testing a singular unit in isolation. That's another reason for integration (and system) testing (Naik & Tripathy 2008, 52). Unit testing can't be overlooked either though, as a unit with too many errors will eat up resources from higher level tests and the causes for issues found on integration tests are harder to identify (Naik & Tripathy 2008, 52).

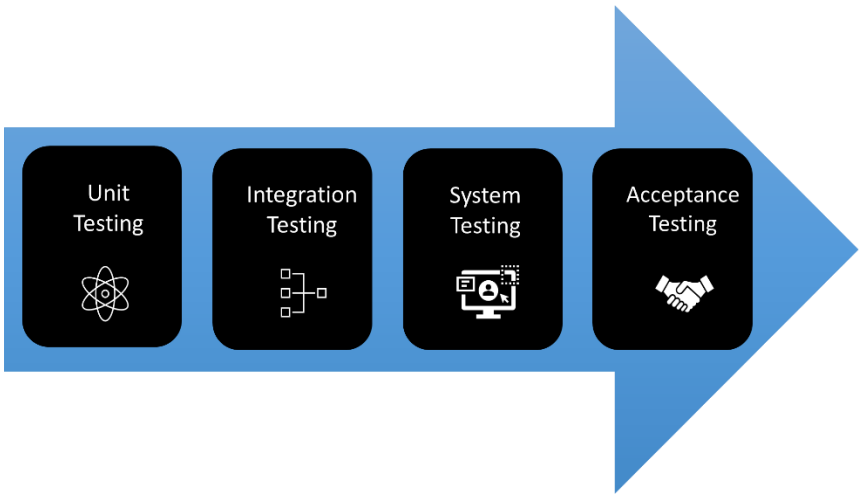

Figure 1. 4 stages of testing a software (adapted from Naik & Tripathy 2008,16)

Integration testing is important for mainly three reasons. Firstly, different units can be done by different programmers, so there will be problems when different types of code interact between each other. Issues between units created by different programmers and even same programmer are extremely common (Naik & Tripathy 2008, 158-159). Secondly, unit tests are done in isolation, controlled. Taking units into the real environment can bring difficulties (Naik & Tripathy 2008, 159). Thirdly, integration testing helps identifying the most error prone units, those that cause the greatest number of failures (Naik & Tripathy 2008, 159).

System testing is done to determine if the implementation matches the requirements by the customer (Naik & Tripathy 2008, 192). Additionally, during system testing, issues can be found that only appear at system level and thus weren't apparent during earlier phases (Lonetti & Marchetti 2018, 8). There are multiple categories of system testing, I focused on the few common ones, what I used in my work. Functionality tests are meant to verify that the system meets the requirements set by the customer (Naik & Tripathy 2008, 198). Included in functionality testing is the GUI-testing. Very closely related to GUI is usability testing, objective of which being evaluating how usable the

system is for the user (Naik & Tripathy 2008, 202). These tests might not be as closely defined in requirements than other functionality, and they are often tested based on best practises in software design. According to Lonetti & Marchetti (2018, 16), usability testing is part of non-functional testing instead of functional testing.

2.2.2 Acceptance testing

Acceptance testing is the last phase of testing before moving into production and launch. The objective is to ensure that system satisfies the acceptance criteria. Essentially, it's used for customer to decide if they will accept the system or not (Naik & Tripathy 2008, 450). It can be perceived as an extension of system testing and not as a different phase (Lonetti & Marchetti 2018, 8). The difference between the two is that system testing focuses on the system meeting the specified requirements and acceptance testing focuses on usability requirements (Lonetti & Marchetti 2018, 8). Jamil & al. (2016, 178) describe acceptance testing as Beta testing, as part of Software Release Lifecycle. Thus, making a distinct difference between acceptance testing and other test levels.

In acceptance testing, an acceptance criteria is crafted. This is done between the supplier and customer to minimize difficulties and misunderstandings during the phase (Naik & Tripathy 2008, 451). This criteria is something that must be met by the system. In the criteria, there are various levels of specifications depending on the project. Naik & Tripathy (2008, 452) mention functional correctness and completeness. This means that the system does what the customer wants it to do. There are various other possible specifications, like accuracy of the provided results, preservation of data and backup to name only a very few (Naik & Tripathy 2008, 454). As the criteria possibilities are vast, customer needs to prioritize their needs to reach an agreement with the vendor (Naik & Tripathy 2008, 461).

Acceptance testing is generally executed by the customers based on acceptance test plan, which is based on acceptance criteria (Naik & Tripathy 2008, 461). Liebel & al. (2013, 22) discover customer involvement being a clear problem in acceptance testing, as almost all their interviewees singled it out. Figure 2 demonstrates acceptance testing, featuring its fundamental contents.

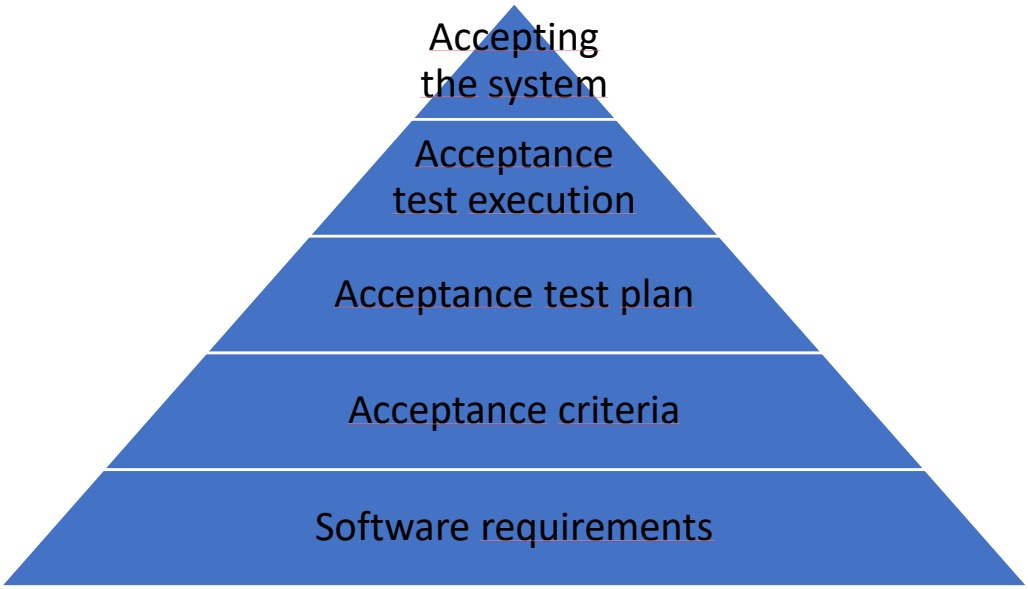

Figure 2. Building blocks of acceptance testing, to reach the acceptance of the system.

The importance of acceptance testing divides opinions. Lonetti & Marchetti (2018, 8) describe acceptance testing more as an extension of system testing and not as a different stage. Generally, though, acceptance testing is thought to be the fourth stage of testing (Naik & Tripathy 2008, 16).

2.2.3 Regression testing

During the full software development lifecycle, regression testing is made. It refers to retesting of a unit, or larger entity (Lonetti & Marchetti 2018, 8). After changes have been made either to fix an error or to add new functionality, there's a possibility that already accepted functionality stops working as a side-effect. This retesting cycle is demonstrated in Figure 3. Regression testing is an area where automation, or some other way of reducing the workload, is effective. If tests are redone after every modification, it will be often and can be very expensive (Lonetti & Marchetti 2018, 8). Because regression testing means redoing tests that have already been accepted, they are generally done at later stages of software development lifecycle. Still, they are done after changing the system, so essentially all the time. If no access to automation, the amount of regression testing will naturally be limited. Still, it can't be fully left to the end of the development, as in that case there would probably be too many previously accepted features that had stopped working.

Figure 3. Regression testing is a cycle of repairing and retesting, as a repair or a new component might break something that had previously been accepted.

2.2.4 Black and white box testing

Black box testing means testing the system from outside perspective. There's no access to code, but rather testing is done via the inputs and outputs of the system (Lonetti & Marchetti 2018, 11). It can be applied to every stage of software development lifecycle (Jamil & al. 2016, 178). It is heavily dependent on the quality and accuracy of the requirements (Lonetti & Marchetti 2018, 13). Conversely, white box testing is done with access to the source code. Because of the need for programming skills for white box testing, it is considered a complex testing process (Jamil & al. 2016, 178). White box testing can be applied to each stage of software development lifecycle (Jamil & al. 2016, 178). In Figure 4, black and white box testing are illustrated, highlighting their unique natures. Gray box testing is a combination of white and black box testing, taking the internal structure into consideration while testing the functionality (Jamil & al. 2016, 178). Lonetti & Marchetti (2018, 14) state that gray box testing focuses on user's point of view instead of the designers.

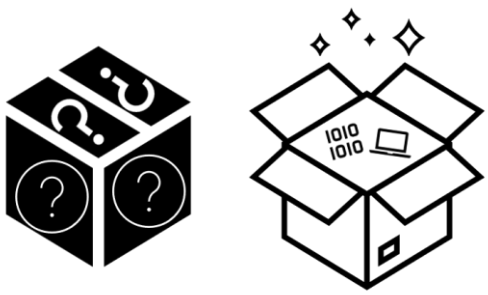

Figure 4. Black and white box testing.

According to Jamil & al. (2016, 178), black box testing is the most widely used testing process worldwide. These methods include more specific techniques within themselves (Lonetti & Marchetti 2018, 11). As a being general terms with such wide usage, they can be considered very important for software development.

Error guessing is a specific way of black-box testing, that is dependent on the experience of the tester. The idea is to guess the areas of system that are more probable to have errors and test those more carefully. Possible error prone areas can be newly added code, code done by junior developers, code with unclear specifications and more (Naik & Tripathy 2008, 255-256).

2.3 Stakeholders

Stakeholders are considered mainly from the perspective of testing, as the main topic of the thesis is testing. Number of stakeholders would be much larger if they were considered from the perspective of the full project. The stakeholders are presented also in Figure 5.

The group working on the project full-time consists of myself and two other people. Before the period covered in the thesis, one worker from the group left the institution. To fill their place, we had another person join our group temporarily, though their employment ends before second user acceptance period. The group has a manager, who also has other responsibilities in the institution. There are other stakeholders in the same institution, whom collaboration is done with. These other stakeholders in the same institution mainly join in the testing for the two user acceptance periods, acting as end user testers. In the project, other stakeholders take part in requirement meetings and related work that’s out of scope for thesis. The small group dedicated to the project consists of my supervisor and my partner. I consider them my partner, as their work responsibilities resemble mine the most. They are the one hired to support the project temporarily. My supervisor assists me throughout the time I’m an employee, and I often ask them insight related to requirements. My supervisor is also our project manager. Most but not all meetings led by me are held between our smaller group.

Vendor acts as a significant stakeholder. Relevant actors from the company providing the system are developers and project manager. Close collaboration is done with developers, as they release new functionality for testing. Further, after testing the functionality, communication with them is done to report potential defects. With project manager, various subjects are discussed, including the content of sprints, requirements, changes, and schedule. To keep the relationship between the vendor and customer healthy, reporting of defects must be done with care and respect. Further, disagreements must be discussed in a civil manner, with a shared goal in mind.

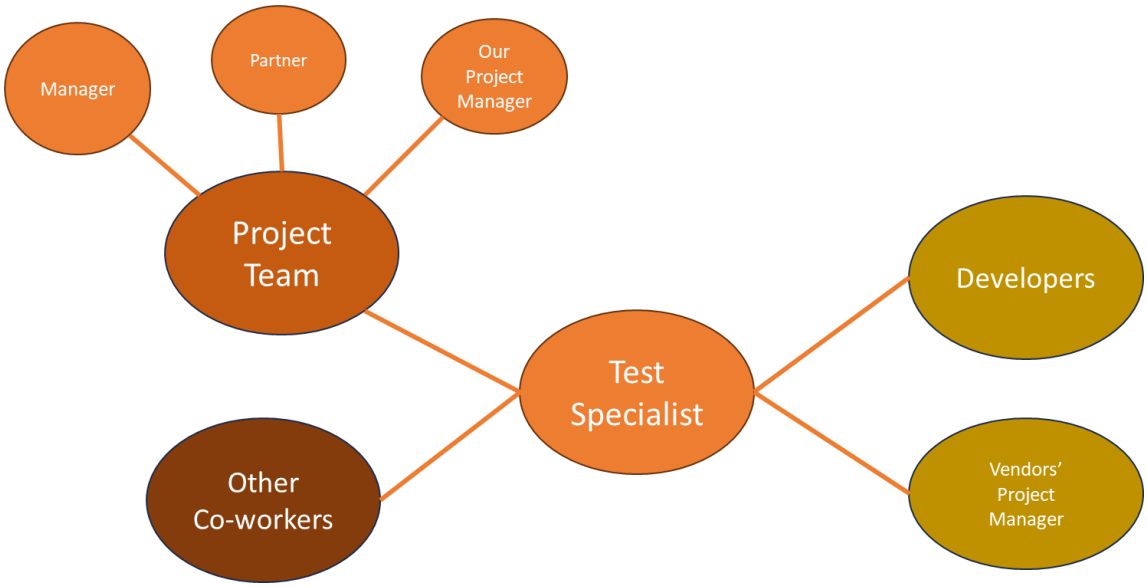

Figure 5. Stakeholders in my work as a test specialist.

2.4 Interaction situations

I work closely with my project team. Especially the project manager is very knowledgeable in the project from many perspectives. They know the different processes that the system would need to perform, as well as project related matters. When uncertain, I ask their opinion on the level of priority for a specific defect. Because my knowledge in the processes of the institution is limited, I often set up my testing results to be confirmed by others in my team. This presents a challenge in my work, as doing such work requires following skills:

- Efficient presentation. The number of issues to cover during these meetings can be significant, making the efficiency of the presentation important.
- Documentation. During presentation is difficult to also document the subjects covered in meeting. Additionally, having clear documentation to use during the presentation makes the meeting go faster and smoother.

I'm leading some type of presentation almost every week, and I need work on both areas. Especially documentation is my weak spot, as I usually just type fast one-word notes or nothing, expecting to remember everything later, leading me to forget most of it.

We have meetings with stakeholders from our vendor often, with varying topics, like sprint planning, requirement specifying or defect explaining. When we, the customer, need to explain a defect, challenges can arise. We need to both understand the requirement and the issue to get the message across well. I don't have a great knowledge in the requirements, as I don't know the

processes of the institution as well as others. I need to gain a deeper understanding of these processes to have easier time leading such meetings.

Further in the project, I gain a bigger responsibility in testing. This leads to me sometimes delegating some testing work to others from our institution. This can bring challenges, as the work will be added on top of the other responsibilities the employee has. I need to consider well who has that additional time to spare as well as enough knowledge in testing. Enough knowledge is essential, as otherwise instructing them might take longer than if I'd done the testing myself.

Sometimes I instruct others less familiar with the system in how to navigate it. Doing such work can be challenging, as it can be difficult sometimes to understand what the other person wants to do. When they don't understand the capabilities of system, they might not understand to ask the right questions. Additionally, if they do ask the right questions, I need to know the answer to them to be able to help them. Or at least I need to know the right person to redirect the question to if I don't know the answer myself.

3 Diary entries

During the diary timeline, two user acceptance testing (UAT) periods are covered. At the end of each week, analysis is done on the diary entries from that week. Before the first UAT-period, general testing duties are discussed. Afterwards, focus shifts towards work related to both UAT-periods.

3.1 Observation week 1

I began writing my diary in the middle of the project, as I had been working in the institution for a good while. For that reason, the first week will reference the week before, even though I haven't written about that week. I had just received a new partner to help with testing last week as well. They had other responsibilities too, but one of the main ones was helping with testing.

Our testing as customers is executed with test cases. The test cases are written by the vendor, providing the service. The vendor in turn, writes the test cases based on the requirements, written together by both parties, but mostly by us customers. With this method, we could give a good idea to the vendors of our needs, and if they didn't understand something, we'd verify the result during testing. In theory, during acceptance testing, there would be minimal amount to be fixed, and us customers could focus on if the system meets the business requirements. Generally, in projects, the customer might not be as closely involved with the production of the system, but this way, risk of having an unacceptable product was minimized. Liebel & al. (2013, 22-23) discover in their case study, that getting customers involved in testing was difficult, and that the involvement wasn't formalized or effective. The aim of higher involvement in our case has indeed been to increase effectiveness and a more precise result.

3.1.1 Monday 17.10.2022

My day started by showing the results of last Friday's work to the supervisor who's off on Fridays. I wanted to clarify my work being right before I'd start closing the stories. The stories were covered by me, my partner, or additionally other testers, who are experts. Their usual tasks aren't testing, but they'll be the ones using the system. They are already getting used to the new system, as well as writing potential notes/errors. They put them to Excel, where me and my partner read them and decided if they were defects, or if they were suggestions. As mentioned in the Introduction, it is common for the developers to handle most of the testing. However, as we, the customers were heavily involved, we could keep steering the production to the right path throughout the whole project.

I read the notes made by these testers and tried figuring out if they were essential, to be considered later or not important. The essentials I showed to my supervisor to ask their opinion. I also helped the other testers by instructing them in quick calls about the testing. On those calls were covered some technical difficulties as well as questions related to the scope of testing now and in in near future. I was able to answer all their questions. Additionally, I had a meeting with one of the developers. We went through an uncertain part of the requirements. I promised to double check the topic internally next day.

Responsibility of functional testing rested mostly on mine and my new partners shoulders, and there was a lot of trust on me to contemplate things like: Does this story need more eyes than just mine? Is this issue relevant, do I need to show it to my supervisor? Does this issue require a meeting? The hierarchy related to work had turned into me figuring out what to discuss with my supervisor, and us figuring out together what next steps would be. With my partner helping me with everyday testing, it worked nicely. Previously it was almost only me, and bit overwhelming. Now, I could share the workload and the stress. Even if I didn't have real responsibility over the project, I worked as if I did, which brought a certain amount of pressure. Now, I had easier time handling that pressure.

3.1.2 Tuesday 18.10.2022

I began my day by presenting the discussion of yesterday with the developer to my supervisor, asking their opinion. With that, we decided on the next step and how to inform the developer. This was tricky, as the developer even after all this back and forth had questions about it and asked one more time with email. It would've been better to have my supervisor in the meeting with me in hindsight, with them having a much clearer picture on the requirements. We probably would have gotten to a result faster, and possibly even to a better one; Now, as I presented the results of our discussion and the agreed solution, my supervisor seemed slightly disappointed in it, and I felt that they would've wanted a better one. What happened was a sum of many things though, not just my lack of expertise. And it's hard to realize early enough, that in this particular issue my supervisor would be needed right away. Either way, the problem was still there, and I'd ask about it from my supervisor once again tomorrow. They were so busy today, that there was no time for more significant discussion.

We also went through remaining tests with my partner doing integration testing, and there was one confusing one that we would return to next day. The test story was quite large, and it was unclear how to proceed at times. This made the testing very difficult, because it was hard to know which part wasn't working. That's why it's essential to have each unit to be integrated be working perfectly before moving on to integration testing. Still, as mentioned in the Discussion section, unit

testing is limited, and doesn't reveal all the issues. So even if unit testing was done perfectly, new issues will still arise. The test case also involved dates, and things being open at certain times of day. Dates are tricky in general, making the story tricky in more ways than one. We trusted in gaining new clarity from being refreshed and rested next day and went home without solving that one case. At this point, system had no other test cases open, so in theory there'd be a lot of time for figuring it tomorrow.

There is one more major part of the system missing from testing, as it's still in development. It's late for about a week now and will probably be open for testing next week. Time is tight already, so hopefully the functionality will get a satisfactory test coverage in the end. However, from our point of view the delay is good news, because it allows us to catch up with other testing. We have cleared up almost all the test cases for now. In general, the state of the project is somewhat worrying, as some functionality is late, and there are many defects written that will need to be fixed also. There's probably around 30 of these defects now. When the developers get to them, there's a possibility of getting overwhelmed due the sheer number of these defects. The issue will spread to management if everything can't be done within time, as prioritization must be made.

3.1.3 Wednesday 19.10.2022

My day consisted of plenty of meetings, but I wasn't much help in them. Still, just attending them broadens the view of what us, the customer wants. After all, I was new to the institution, and needed a deeper understanding of the processes done by the experts.

We finished the last open test cases with my partner. There was one tricky one from yesterday, where we had to play with start and end date to get the wanted result. It had to be possible to have multiple reports from same form, starting on different dates. This led to having to test if expired reports were still visible to the user. The way to test was throughout multiple days. Testing this, we concluded that there were indeed problems, and the older reports went hidden after the expiration date, which was unwanted. It was unclear if the error was because of us testers or because of system. Either way, the forms were hidden, so we couldn't solve the issue or test further. With us customers doing the tests without access to code, after identifying the issue, it was up to the vendor to figure out why the issue was happening. In other words, our job was done.

At the moment, all the tests had to be accepted with low standards due the time limitations. Any issue that prevented the flow of the system, was pushed for later date. Because our product had inflexible deadline, we had to prioritize heavily to get a minimal viable product on time. The defects that were pushed later kept piling on, and there was a risk that there wouldn't be enough time for fixing everything later. Still, launching the minimal viable product was our main goal now.

3.1.4 Thursday 20.10.2022

New test cases arrived today, for a special kind of form and added pages to be tested for normal forms. My work was functional testing, and I worked with a partner. Process seemed very straightforward initially, to compare the requirements with product. It wasn't as cut and try in reality though, as the requirements in the Excel were slightly unclear. We had a question or two from every page and needed to show it to supervisor to clarify if some of the solutions were within the requirements or not. Unfortunately, we couldn't accept much ourselves because the requirements themselves were a bit unclear as well. This wasted us a lot of time, because we had to make a new form alongside the tested form, to compare the results and figure out if everything worked as intended. The intended original functionality was unclear, so our tests gave a result, but we weren't sure if the result was correct or not. If we succeeded or not would be found out tomorrow, as the supervisor was too busy to talk about it today.

In earlier sprints, the requirement Excel had been more refined, which made the testing simpler. I couldn't accept the cases all by myself most of the time back then either, but the difference was still significant on the percentage of cases I could do basically by myself. I was able to figure out the errors, and showcase them to my supervisor, and they would agree on it being an error 95% of the time. Now, the requirements weren't as clear, so I ended up showing things that I merely thought to be sketchy or wrong. Agreeing percentage has dropped to about 60%, leading to time-wasting twice during my work. First, when I found out errors, I had to use more time in doing guessing and research using excels or earlier made forms, trying to figure out if the functionality shown was correct. Second, I presented more errors that weren't actually errors. Interestingly, the number of errors as a whole didn't increase, the increase came from my misunderstandings. This would mean that the developers were able to decipher the requirements better than me despite their lower level of accuracy. This was surprising because I did need more time now for understanding them. One would think that they'd make more mistakes as well.

The familiarity with the project has increased now, so developers don't make simple mistakes anymore as much. For example, in today's test for the first time, a textbox included validation from the get-go, to not allow other input besides numbers. Before, there was no validation when I tested them. This resulted in me reporting small things to be fixed, that might've been expected to be covered in unit tests. Unit tests are generally done by developers, to minimize issues during integration testing. Now as I tested the functionality of many units, it felt more like integration testing than before, as individual units worked as intended.

3.1.5 Friday 21.10.2022

I presented all the test notes to supervisor as they'd been busy yesterday. Based on that meeting, I was able to record the issues to the system. Few of the notes were registered as a defect, to be fixed later. One of the notes we commented directly into the case story, to be resolved immediately. One of the notes required an email to our specialists, to verify if the requirements were up to date, or if there was need for supplementing. Personally, I was able to improve on my preparation towards presenting the findings. I was able to find and replicate the errors very fast to my supervisor. This allowed for the presentation to go very smoothly, and to go through all the points I had made. At one point I was not able to understand my own notes that I'd written yesterday, so I still had room for improvement. I tend to make fast notes, that later become difficult to decipher. I was putting effort into making them readable, but I relied too much on my memory still. My thinking went that if I had enough bullet points, I'd remember my earlier thinking process. This sometimes backfired, as at times I had to waste a few minutes to remind myself my earlier thought process. This was a fault in myself I recognised and intended to improve on. I managed to remember my thinking process on all the notes except for one.

Small problem right now in my work was documentation. The number of stories to go through was about 10-15 most days, and there was something to say from most of them. Management of different documentation was difficult. I couldn't forward the notes to developers immediately, as I had to present most of them first to my supervisor. It was a slow process, as I had to have them not only for the presentation, but also for the reporting process. I needed to make sure I didn't lose any of them before they made their way to developers. Right now, I put them in a notebook naming them by the defect code. I made codes for their status:

> ? meant to ask about the defect.
>
> ! meant to record the defect to the developers. Usually changed from question mark after asking.
>
> VV meant finished. I put this code after deciding the defect to not be recorder, or after recording it to developers, to keep track pf what I'd done and hadn't done.

The problems arose when something was discovered later related to already recorded defect. In this case, I put new findings to different place, making the structure unorganized. This could've been avoided with a more disciplined structure. I needed to remember, that it helped me immensely on later stages of testing process to keep the documentation perfectly structured.

3.1.6 Week Analysis 1

The work week consisted mostly of functionality testing. There were a few issues with the requirements delivered, which meant that the functionality to be tested wasn't completely clear. Because of this, in addition to functionality testing was to re-clarify the functionality wanted. It's quite common for issues to arise from requirements, including disagreement of interpretation and making changes to requirements (Naik & Tripathy 2008, 322). The system was quite a complicated one, and only seemed to be getting more complicated as time went on. One development part got delayed for a week, taking double the time from what was initially expected. The acceptance testing was planned to be started at 31.10, which would be a week away, but was pushed a week further. There were parts other than the just mentioned functionality, that had taken longer than initially planned, which is why the acceptance testing would start later. The launch couldn't be delayed though, because of the strict deadlines put for this project. This meant that the acceptance testing time would be shorter. For the upcoming few weeks, it would be crucial to do clinical but fast work, to achieve a working result on time.

For us testers, this mostly meant to think critically about every issue discovered. Was this error crucial? Had it to be fixed now? It became increasingly more important to come up with the right timeline for every defect, to end up with a viable product in the end. Of course, this kind of prioritisation should always be in the mind of a tester, but now even more so. According to Naik & Tripathy (2018, 371), it's a big problem for developers if testers report critical defects too late, as they won't have enough time to fix the issues. The decision of the prioritization wasn't in the hands of the testers alone. The development team argued on some of the reported incidents, disagreeing them to be time sensitive. When this happened, our team rethought if it indeed was something that had to be fixed now or not. If it was, the communication continued with the developers. This communication could be done by me, or my supervisor, depending on the nature of the problem. Sometimes it was hard to know if the supervisor was needed or not. On Tuesday I held the meeting by myself, and we failed to come to a satisfactory conclusion. Things might have been different if my supervisor had attended that one. It had to be remembered though, that the schedule was unforgiving, and the decisions might've been unsatisfactory for that reason alone. Either way, I wanted to improve on my foresight, to figure if I could handle a certain meeting or if I'd set up one for my supervisor instead. It is an important skill to have a realistic understanding of one's own capabilities and to know when help is needed. As I failed to measure this properly, more time was wasted on extra meetings later.

One technique to prioritize defects is to use tiers. With three tiers, the high priority ones are marked red, medium priority ones are marked orange, and low priority ones are marked green (Naik &

Tripathy). Naturally, the ones marked red would be delt with first. In our reporting system, as presented in Figure 6, four tiers were used with the following meanings: Red marking meant that the defect had to be fixed immediately, it was a first-tier problem. These types of defects meant that something was broken, or something prevented other functionality to be tested, or something wasn't at all what was in the requirements. Orange marking meant, that the issue needed to be fixed during that sprint but wasn't as time sensitive. It was a second-tier problem. Third-tier problem, color-coded orange, meant that the issue needed to be solved eventually, when there'd be time. It didn't need to be fixed even during that sprint but was still saved in the system for keeping track of defects. The fourth tier, green, meant a cosmetic change, or an issue related to language. The system was translated to Finnish and Swedish from English, and sometimes parts were still in the wrong language. Finally, for all four tiers, we needed to specify if the issue was an added functionality or not. If it was something, that wasn't part of the requirements, it would be a new functionality. This was important to specify, as it'd cost extra for us, the customer. Still, if we realized something to be fundamental to the system, that we failed to mention in the requirements, we added a defect including completely new functionality. Most defects were not a new functionality. A few were though, making this feature necessary.

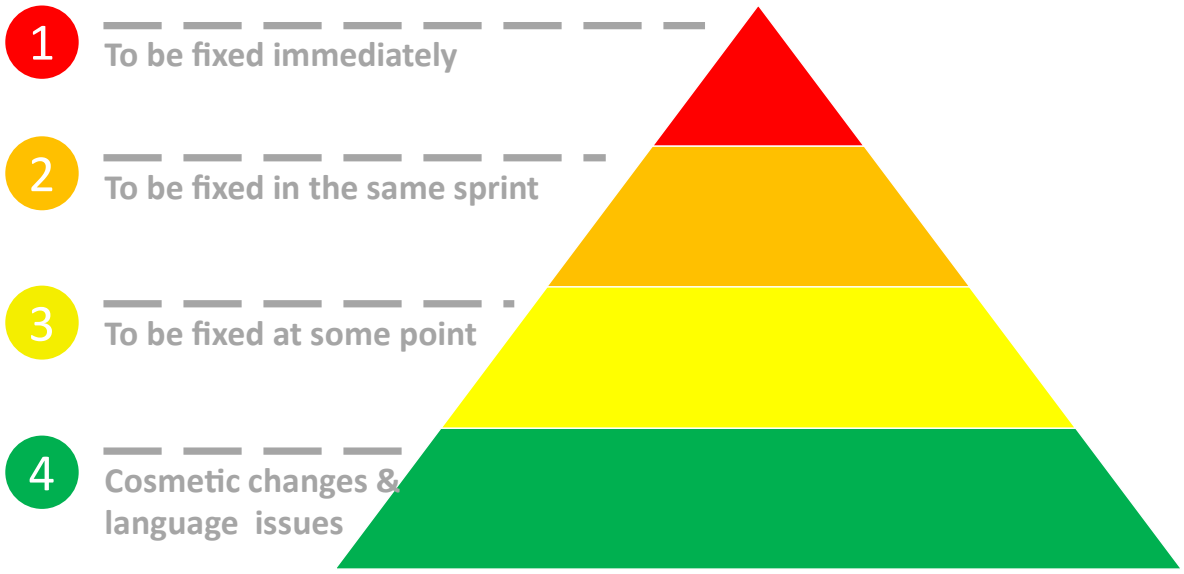

Figure 6. Four tiers of priority used in our project.

Time management was a key element in all our work at the moment, and it dictated our decision making as well. At school I had learned to report everything and was taught, that all discovered errors must be dealt with before production. It was quite precise, and small details mattered. This was a fantastic prerequisite for working, even though work has been proven to be slightly different. I've had to use much more of my own judgement on if the issue found is indeed crucial, or if should be ignored, or something in between. Like Hambling, Morgan, Samaroo, Thompson & Williams

(2008, 164) explain, sometimes tests need to be reprioritized if an identified project risk occurs. As our software is running late, we need to a lot of this work.

I was aware that the development in real projects can take time, and that big projects tend to be late. Now, as some parts of the development are over a week late, and some have been pushed to the second part of development altogether, it really has hit me how much that can be true. Initially, the project plan stated that the production would be ready at the end of year. Now, the production has been split, so at least the most important functionality would fit into that schedule. With those prioritizations, the production timeline has been managed. In the initial specification phase, the belief was for the development to take less time. But it turned out the software platform used didn't immediately match the needs of our institution and needed to be modified. These modifications proved to be difficult, thus stretching the time used. Requirements have been found lacking at times, and production has been found to take too long at times. These are the main reasons for delays, and to an extent, they might have been preventable. On our side, we've paid additional attention to the requirements, but even still, issues arise. We're trying to keep them at minimum though.

3.2 Observation week 2

In the second week, Monday to Friday is covered. Nature of work is mostly black box testing and evaluation of defects, ranking them by their importance.

3.2.1 Monday 24.10.2022

On Friday we set up a date-sensitive test that would be testable the next day. Days have passed, and now on Monday we could perform the test. The functionality wanted was to have two forms. Earlier one had an expiration date for Friday, thus being already expired today on Monday. The following form was meant to start today. We needed to test if made changes would carry over from old form to the new form, and if not, what would and what wouldn't carry over? Unfortunately, we could not test this, as the form behaved unexpectedly even earlier. It seemed like the new form, when trying to be opened could not open, but froze the screen instead. We were using a special kind of form for this test. Now, we had to do the test again, with the special forms and normal forms, to figure out if the problem lied in using the special type of form. Additionally, we needed to do one where changes were made, and one where there were no changes. We had to pinpoint the problem, so we needed to cover all the main possibilities where things could go wrong. My programming background helped me a lot in this, as I knew the different variables to consider where the issue might lie. As this was date related test as well though, we had to wait for results tomorrow, as we needed to restart from the beginning. Our hope was that some of these different test

forms would work, and some didn't, so we'd know where the problem was. If they all froze, then the problem would be more general, and after fixing it, special forms would need to be retested.

As we were making the test forms, we happened to take a closer look at the other side of form applications, the side of handler who receives the application. The acceptance testing is set to begin in a week, and there haven't been any test cases about that side. All the UI tests and modifications have been made in the applicant's side, and they don't carry over to the other side, as they are different systems. As we went over the other side, we found a few critical issues that had to be acknowledged. Biggest one, that prevented the work of handler to an extent was a part of table not being fully visible with no way to be turned completely visible. This text was needed for valuation of end users work, so it was a severe error. The table was more advanced in the applicant's side, having a button for expanding the table for better visibility. This was missing in the other side, and the text was mostly unreachable outside the screen. That wasn't the only issue, but it was the one that was the most critical. It had to be fixed before the acceptance period, so I sent an email to the developers about it.

3.2.2 Tuesday 25.10.2022

Day consisted of meetings and small amounts of testing. I also used a lot of time for working with Excel, helping my co-worker, and fixing customers' broken Excel pages.

The acceptance testing would be very soon, so we had to rank the defects based on importance using the four tiers earlier mentioned, as everything couldn't be completed by the AT period. I continued that work. There were 35 defects, and I had to rank them from 1 to 4 based on their level of importance. Days available to do these fixes aren't too many, so realistically there can't be too many ones that will be fixed. Still, I had to give five issues the highest rank. Things that have been in development were late also at the moment. These delays would give some uncertainty to the acceptance period, but we have been in a good communication between the customer and the vendor about the most important features and defects that had to be done by the deadline. So, even if some things, even many of them, get delayed over the deadline, we still have a good chance of launching a minimum viable product on time. Then, the rest can be done with a new deadline, as those features will not be used immediately on launch by the customer.

3.2.3 Wednesday 26.10.2022

I finished with the rest of available defects. I rated them with the usual scale. Tomorrow, my plan would be to present my ratings to my supervisor, and we'd go from there. We still hadn't gotten the test cases about the new functionality. This meant to prepare ourselves for a quick and intensive testing period when they'd arrive. The functionality was meant to be ready for publishing at the end

of this week, and as it wasn't ready for testing today on Wednesday, reaching the initial deadline seemed unlikely.

I communicated with a developer regarding a defect I had reported previously. I asked about a new functionality that wasn't clearly stated in the requirements. It was about continuity between forms. How the data from old form presented itself in the new one. This was a required functionality, but it was failed to be described in requirements in a satisfactory manner. It's not always easy to think from the perspective of a computer. The need was to include the input from old form to the new one. Naturally, data doesn't transfer over without programming it, but we failed to think about it during defining requirements. This wasn't the first time something like this happened, and considering the size of this project, I'm surprised it had happened as little as it had. The requirements and the functionality were more complex than I had ever seen. Either way, I provided the updated requirements to developers after discussing them with my colleagues.

During the communication I also discovered the reason for Monday's bug. I'm referring to the bug where the second form freezes after the previous one is over deadline. The reason this happened was because of fixes made in another part of the form, affecting this as well. This was what the developers thought at least and would need to be tested after the other part had finished development. Testing would take two days again, and we didn't receive permission to start it today yet. These kinds of problems come up during regression testing. As explained in the testing discussion, it's possible after adding new functionality, that an already accepted part breaks.

3.2.4 Thursday 27.10.2022

We went through my initial rankings of defects with my supervisor. We mostly agreed with my rankings, but my supervisor being in management of the project, wanted to push a few more defects for lower priority than me. They were concerned if the developers could do the defects on time and didn't want to overwhelm them. I agreed, and based on our meeting, sent the defects forward.

The rest of day went working with Excels and other work not in scope for my thesis.

3.2.5 Friday 28.10.2022

I worked alone today, which meant to plan out the day by myself and document well, so the team could efficiently use my findings on Monday. The functionality I've been written about throughout the week that was supposed to be finished this day, had one part of it released for testing. I didn't touch it yet, as that testing would be much easier done by someone with expert knowledge. Today old test cases were sent back to us with new repairs. These cases had small differences but had a lot in common as well. I had already tested one of them, so I now worked on the rest. I had to

make plenty of notes, as these too required more than one form, separated by time. I made many tests that would be finished on Monday when the new form would open. The previous issue regarding time was now solved, so I looked for other potential issues with the same theme in mind. Being wiser from before, I documented my work much better to my own notes, and the notes would help me complete my tests on Monday, as well as to present the issues to my co-workers. My plan will be to complete these test cases fast, and it should be possible provided I can put others up to speed quickly. The functionality with delays should be ready for testing soon, potentially on Monday, which is why other matters should be finalised by then.

I also did some Excel work. Many people direct their Excel-related questions to me now, so I've usually done something small for someone with Excel almost every day.

3.2.6 Week Analysis 2

The biggest logistical issue right now has been time restrictions on the forms. Because of them, the otherwise simple testing has taken days to complete. Of course, other things could be done when waiting, so the problem hasn't been as critical as it could be. If a defect prevents the execution of a test case, it's called a blocked defect (Naik & Tripathy 2008, 364). If critical things to be tested were behind the time restriction, thus blocked, situation would be worse. Still, a solution should be found, something that takes away the restriction yet keeps the functionality same. This is for testing purposes of course, and for the final product the restriction needs to be kept. Regarding the issue, I will propose to have the time restrictions removed from the test area, thus having separate functionality for development and testing side. The matter is not very complicated, so it should be possible to keep track of the differences between the respective sides. Generally, it's not a good idea to have differences between testing and development environments, but the value exceeds the risk in this case.

The failure of setting up concise requirements are very common reason for projects to delay or even fail (Ghule 2014, 546). Knowing it isn't always enough to prevent it. Our project was simply so huge, that thinking about everything was an incredibly tough task. We've found ourselves with few cases where the requirements were unclear or lacking. This resulted in additional work for both parties, as we've needed to supplement the requirements, and developers have needed to add new things, or redo old things based on the corrected requirement documents.

There are many sections of the functionality. For now, we've been testing only a relatively small part of the whole. This is mainly because not everything will be made at the same time but rather the scope of each sprint is precise and small. In an agile project, the idea is to constantly valuate how the system is shaping by testing every developed functionality in their respective sprint. This

methodology is called an iterative approach (Edeki 2015, 22). With iterative approach, it's easier to keep the project going towards wanted direction, and not have a situation where a system is half broken when the testing begins. Still, because we've been focusing on the precise parts, we did forget to think about some other important things. The other parts didn't have stories made of them, but they'll still need to be functional for the upcoming acceptance testing. Liebel & Al. (2013, 22) discover in their case study that some companies don't have enough time for testing. It can be difficult to have enough time, as developing the functionality is very important as well, and might take all time that's available. That's why testers are often a separated group to developers, dedicated for testing (Naik & Tripathy 2008, 497). For us too, our test group is separated from developers, as we are a different institution from the developers. Vendor performs tests as well, but us having a dedicated team for testing gives a higher chance of good test coverage. Our side doesn't make the test cases though, which might explain why some of the functionality wasn't written about. If we'd left this uncovered functionality to the acceptance period, developers would have had less time for the fix. This taught me an important lesson: No functionality is simple enough that it can be left unchecked, and it's possible that everyone forgets to think about something in a big project. It's better to double check, if possible. Of course, time limitations open room for mistakes.

One of my last week's issues was leading meetings. This week I made extra effort into my own note making with that in mind. Last week I used one-word reminders on my notes, this week I instead wrote full sentences. Like mentioned before, some of my testing continued to next day or even longer, and I found that small change quite useful. I still had to take a small time when reading my old notes, to figure out my earlier thoughts and insights, but now there were no cases where I completely dropped something, because I couldn't understand it. That had happened last week. When I was presenting, I had dismissed the points I couldn't quickly enough remember. I didn't have too many meetings this week, but I was still able to make use of my improved notes on my work, when I registered my findings to the developers. Instead of reading one word and then replicating the error on the system, I could just write my earlier note to the developers as a report instead, not doing the whole process twice. There is also the matter of importance I need to ponder before sending. That's why having the notes written down in a clearer way is helpful.

Large amount of system testing techniques revolve around prioritization (Liebel & Al 2013, 18). Rating of defects based on their importance continued this week, performing prioritisation. I ended up giving quite a few level 2s, which means to be fixed before user acceptance period begins. I had to consider the short time but also consider if there were cases which could be handled with additional instructions for the end user or if they needed to be repaired by developers instead. If repairing was to be done, I'd then need to think about the timeline goal for the repairs. I mainly did this by myself but also talked with our worker responsible for creating the instructions. I asked them

whether they could make clear instructions on how to get around something that would not be fixed, or if it had to be repaired instead.

3.3 Observation Week 3, week before first UAT-period

As the first UAT-period is arriving next week, this week is preparation for that. With preparation comes meetings, and this week has plenty of them. Like last week, black-box testing is also performed.

3.3.1 Monday 31.10.2022

I began the week by presenting my findings from the previous week to my colleagues. I showed the time-dependant tests that were opened today, where I'd discovered that some things didn't work as intended. One form had its content cleared upon turning a button from yes to no on a previous page. This button is initially turned to no. So, on the second form, the form got deleted every time as the button was turned to no by default. There were also some other usability issues, like the fact that there was no information on this behaviour, or warning, or possibility to undo it. I presented this among other things, taking advantage of my more readable notes. My partner pointed out some new things during my presentation, of which I wrote quick notes, going back to my bad habits of writing hard to understand notes. I had some way to go still, but because of the good notes I had written last Friday, and because I had prepared by myself a little before the presentation, I could be much faster and smoother with it. My partner even mentioned that I was on top of everything on the meeting. I wouldn't go as far myself, but it was clear that my behaviour change brought improvements.

I used my day for continuing tests, testing units and their relationship between each other. I made sure everything matched the requirements. I also did things that the system shouldn't allow and checked if they were allowed. These included adding wrong kind of data for example. This type of testing falls under "error guessing," which I covered in my testing discussion. I relied on my expertise in knowing parts of system most prone for errors and paid more attention on them. My testing was standard, except for the time related tests mentioned earlier.

I also led a meeting concerning testing in the upcoming days. The tests related to calculations had arrived, and they included many different variables to cover. We had to share the work, and in the meeting, we decided how to go about the sharing as well as informing the others on their tasks. Afterwards I communicated our decisions via email. Time would be on the essence, so hopefully everything will be covered now.

I also helped my co-worker on a test-related issue on a quick meeting and did more Excel tasks that were forwarded to me. The meeting went well, and it helped blossom a few ideas for improving the testing experience in future. A simple thing we had forgotten was to share the requirements Excel with the other testers. They were the experts, and it slipped our minds that they would benefit from documentation as well.

3.3.2 Tuesday 01.11.2022

I was at the office today, and I had a lot of meetings. My day was very scattered, with meetings and one demo in the morning. It was difficult to do my tasks, as the meetings kept interrupting the workflow. Only on the afternoon I was able to work on the tests because my morning went with meetings and some Excel tasks I was assigned to. Thankfully, I had written yesterday my tasks I needed to do for today, so even if I only got to them late and tired, I was still able to do them in a satisfying manner.

I did testing on the same forms as previous days. For a certain period, there is always only one form to fill, and after that there is a possibility of another one, with the old data. There can be more than two of these forms, so I tested a third form with the old data from first form and added data from second form. I wanted to see if the data carried over correctly from every form. Sometimes, when I'd send a report to developers about an issue that was related to more than one section, they might forget to fix the others, or add the repaired functionality to the others. They might only add the new code for the very thing asked. This is what happened now, as some of the continuity functionality that was required to be included worked on second form but didn't work on the third form. Something like this can be easy to overlook, and it can stay in the code for a long time. It might even be only noticed when the end user is making a third form. That's why it was vital to remember to take these rarer cases into consideration when testing. Thankfully, the issues were discovered now by us, and not by the end user later.

On another spot in the form was a question, with a yes or no answer. "Yes" would open a table where the user could fill data in. "No" would hide that table and clear it at the same time. The default answer was no. It was discovered that the combination of these functionalities led to the second form having the data deleted, as the button turned to the default answer no, deleting the table at the same time. This data was supposed to last between forms, but the connection between the table and the button was not considered when the new form was opened. Similar problem had been discovered yesterday as well.

Even if the time was short for testing today, some critical discoveries were found. There are a lot of things to consider when working with continuity and change of time.

3.3.3 Wednesday 02.11.2022

The theme of endless meetings continued today, as there was a total of 5 meetings today. This made doing my actual work exceedingly difficult. How to work in an efficient manner while having meetings that constantly disrupt the workflow? I'd have to try find a solution for that question.

One of the meetings was about a few test cases, and we discussed functionality in the system. It was basic testing, comparing the requirements to end product. Results were unclear however, which is why we looked at it together. Even if I didn't get much meaningful work done, the meetings laid a solid groundwork for tomorrow's work. I should get the rest of open test cases tested easily now. I will write about it in more detail tomorrow as well, as I didn't discover much today. The biggest problem right now was that the system had a few places where there were multiple possible paths. These paths were so different, that they needed own separate forms. All different forms had to be separately tested. There's an additional variable for each form with 3 paths, tripling the amount of possible unique forms from the earlier amount. Making a form takes its own time, and in this part of testing, we needed to create forms for all the variables. The testing itself wasn't simple either, as there was a lot of calculation and formulas involved. All this combined, the testing took a lot of time and care, and it got confusing fast. Knowing all this, I hoped I could stay on top of it tomorrow. I'd start the day with some planning, so I wouldn't get confused on the way.

3.3.4 Thursday 03.11.2022

The testing proceeded forward well, but the time was running short. There were 14 stories, that I'd have to come to some conclusions with, as next week the acceptance testing would start. Today went smoothly though, and I was able to complete tasks in a good order without wasting much time. I first gathered all the issues I found from testing and had a meeting with my supervisor about them when they had the time. While waiting on that, I prepared forms for more testing later and did testing on other parts of system. My notes were understandable enough that I could present everything in 20 minutes and could proceed to reporting the issues forward.

I had written about an urgent issue few days ago to developers that I hadn't gotten a response yet. It wasn't a surprise; the receivers were incredibly busy right now of course. Either way, I filed the same error as a defect now as well, ranked highest priority, so it would be fixed as fast as possible. These kinds of things just flying around had to be dealt with now when there was still time. I hadn't written them down as to-do list or anything, which created a risk that something would be forgotten. In hindsight, I should have written a to-do list. It was another place for improvement for me. A to-do list for tasks that were not to be done immediately, so I'd remember to do them when the time came.

My tests today fell under integration testing category, as I checked how parts of system interacted together. I needed to do some validation tests also. There was a box that included validation checking for correct variable, and another box that should have, did not. That probably meant, that the developers were thinking about validation more now, but still forgot something. According to Hambling & al. (2015, 147), developers can lose their sense of responsibility as they know they have a separate test team, that will find the bugs anyway. It might have been the case in our project, though the reason might've been simply the priority of development over testing for them. Either way, that kind of testing was very simple, but the testing of functionality was much harder. The requirements Excel we used was difficult to understand at times, making the comparing it to the results slow. I usually needed verification from higher ups as well. That's why I held the meeting with my supervisor, and we were able to come to conclusions together. Tactic of first doing myself and verifying later gave me results. Doing verification even earlier would probably have sped up the process even more, but it was not possible due to availability issues.

3.3.5 Friday 04.11.2022

It was the last day before acceptance testing, and some issues had risen. Because of the tight schedule, we previously had to write some issues as defects that would be fixed when there'd be time. Now the extended time for fixing had bitten us though, as some functionality was dependent on the previous parts working properly, and they were not yet fixed. For example, part of data was not moving from one page to another. The second page's data should move to a completely new form, the one that we would test now. But, because the connection was broken, we couldn't test what we need to test. As explained in the last weeks Week Analysis, this phenomenon is called blocked defect, and is a more serious issue, and a high priority to solve. Another problem was in the functionality that would be required for the forms. Because of the bureaucracy, in between the forms had to be a waiting time of 30 days before the next one could be filled. There were multiple tests that had to be ran on the second form, mainly to test how the data moved from one form to another. Today, the requirement of needing to wait 30 days was added, which made it practically impossible to properly test some functionality. It was a tricky situation, as it was a needed functionality, but it shouldn't have been added yet, as our testing couldn't be done properly now. I communicated this to developers, and hopefully it would get fixed by Monday.

I hoped the critical issues would be fixed very soon, so during acceptance testing everything that should be tested, could be. It would be very bad if the environment itself would prevent the testing. If they were not fixed on Monday, I probably will have to mention it to my supervisor, to at least let them know about the parts that we wouldn't be able test at first. Again, planning ahead will be useful, and knowing what to tell the supervisor ahead of time.

The other testing went smoothly. I did variations of previous testing with different forms. The content of the form was similar, but different enough to warrant separate test cases. It was similar enough to have the content mostly the same though, making the testing process straightforward. Regarding this, the hardest work was already done previously on the first form, and now these slight variations went over fast. That's not to say there was zero issues, they were just much fewer. Again, I naturally reported the issues I did find and sent them to developer team.

3.3.6 Week Analysis 3

According to Ghule (2014, 547), one of the main risks in developing software is an unrealistic timeline. In our project, keeping to timeline has been an issue as well, as some development has taken an unexpectedly long, and we've had to push some development for next year. Edeki (2015, 23) points out that using agile methodology in development allows for the risk of falling behind to present itself very fast, due to iterative nature of agile. If this happens, adjustments can be made before it'd be too late and the whole project would have to be cancelled. Instead, with prioritization and moving some parts to be developed later, project timeline can be adjusted to reach a satisfactory outcome.

Last week's problem with time restrictions were only fixed for a week, and now they have returned. It is unfortunate, as ideally, we would have tested everything related to second and third form before activating the time restrictions about them. But because of last week's problems, we didn't have enough time to do that. Now, we must temporarily stop functionality to be able to test it. To prevent these kinds of problems, our communication needs to be clearer between testers and developers, as this should have been preventable.

My rating for the importance of our defects from previous week was successful, and it was mostly accepted immediately by my co-workers. My preparation for it paid off, as I had good and understandable notes and sensible order with the defects. My work is improving with small quality of life changes right now. Having an excel for other people's findings helps me by seeing if there is something I hadn't found myself and it helps even more indirectly because others can check from there instead of asking me if something has already been noted. Additionally, me being more prepared in meetings have kept their length much shorter, and I have been able to go through everything I've needed to.

Testing I was doing this week required quite a lot of time management skills. We have many different types of forms, that still include similar contents within them. If I only did them one by one, it would take a lot of time, and it would be difficult to keep track of the problems of a certain form. But because I did many of them at once, I remembered the issues of a certain form when doing the

next form straight after. This allowed me to pay additional attention if the next form had same issues or not. While the first form was waiting to be accepted, I could move on to the next one, thus saving time as well as staying on top of the requirements to look out for. As discussed on Wednesday 02.11.2022, the different form possibilities had been increased to a significant number. Doing a lot of slightly similar work would be more effective using automation. As explained in the Testing discussion section, repetitive testing is very suitable for automation. Using automation is not in the scope of this thesis, but for future an important skill to learn.

The acceptance testing is starting next week, and it's my first one with a bigger project. We have shared the duties ahead of time with the larger team, the one that's not usually part of testing. For the original testers, we have a lot of testing ahead, as well as managing others. We are responsible for getting the issues found by other testers to developers in addition to doing the largest amount of testing. I had some issues this week with having meetings disrupting my workflow, and I'm slightly worried that this different type of work I have next week will have the same disruptive effect. Other risk factor for acceptance period is the amount of testing needed. Some of the functionality has been late, and its testing will be left for the period that should be used for other testing. Meaning that something will be not tested properly, either the full scope of acceptance testing or the functionality yet to be added. There shouldn't be any more functionality testing any more in accepting period, but there will have to be because of the delays. According to Hambling & al. (2015, 51), acceptance testing is used to determine if the system meets the customers business needs It shouldn't be a time for findings bugs anymore, but to confirm the big picture being correct. Naik & Tripathy (2008, 462) state that during acceptance testing period, emphasis is on the system working according to customer's expectation, rather than passing tests. Furthermore, the system should have already passed such tests earlier during system-level testing. This has been the case for us as well, and we are going into the period with similar mindset. However, due the delays, we've had to include a few sets of features that haven't gone through a satisfactory amount of testing.

3.4 Observation Week 4, first week of first UAT-period

As the first UAT-period begins, this week will cover the involvement of larger group performing testing. Focus will be on getting everyone familiar with the new system and confirming the business requirements of the system.

3.4.1 Monday 07.11.2022

Today was the first day of the acceptance testing, which meant a lot of preparation meetings. We had an internal meeting with our smaller testing team, where we went through the last days of testing, and prepared for the meeting with others in our group. Then we had the meeting where we

answered all the questions other people had about testing. It wasn't just the smaller group for the testing now, so we'd hold meetings for the others daily for questions and assistance. Finally, we had a meeting with the developers, where we discussed everything that was in scope for the testing, as we had split the development into two parts, with the second half to do be done next year. Additionally, we went over what was currently in development, ready to be tested soon. We also discussed the acceptance criteria, minimum results acceptable for us to proceed to production.

I couldn't do a lot of actual work today, as it was all planning. The acceptance testing will take two weeks, so it was important to have solid plan for it. We will do exploratory testing, system testing and user acceptance testing. We made an excel for notes that everyone could fill, and then our smaller team would filter them and notify the developers. We'll also evaluate the significance of the reported issue from 1-4 with 1 being critical and 4 being an improvement suggestion. To help our whole team in the testing process, we created general instructions for the test period.

The issue mentioned last Friday about the 30-day wait between forms had been now fixed, and I was able to start the time related tests again.

3.4.2 Tuesday 08.11.2022

The managing of work during the acceptance testing was proving to be difficult. There was so much to cover, and most of the time went into answering test-related questions and meetings. We had multiple meetings a day with different people about testing. One of them was about our own internal findings, one of them was going through the findings of others and one of them was answering any questions other testers might've had. These three will reoccur every day for UAT-period, and today we had one additional meeting with the developers also. All these meetings shrunk our own testing time significantly, as ideally, we would test the whole day. Our testing included techniques others didn't use, like the intention of breaking the system. It's not to say our testing was most valuable though, as the experts were still the ones who user acceptance testing period was for, as they'd know what was required from the system best. Still, our testing was also valuable for the end result, and we needed to find the time for it somehow.

I was able to do some testing still, testing a new part of form for the first time. It had a lot of issues, like having a table pop up when pressed no, instead of yes. So, the functionality didn't work as intended. Other issues I noted were an un-editable box that should have been editable and a table that had a duplicate of itself. Small bugs like these wouldn't be expected to be found on UAT-period anymore. Mistakes happen though, and it was still better now than later.

I found myself needing more time for testing than I had today. I'd have to think of a way to get more time of the day for it. I'll probably have to prioritize testing over meetings and cancel some of them

in the future and not help my co-workers outside of our help-meeting so I could centralize that work and keep it contained so it doesn't spread to the whole day.

3.4.3 Thursday 10.11.2022

Because of personal matters, I took one day off on Wednesday, and came back for Thursday. My initial trainee contract ended this week and another, 80% work week started. I will have Fridays off now, starting from this week.

Returning to work, I first caught up with what had happened yesterday. There had been made many new issues on the system. The testing was going well, but there were some worries. We were finding a significant number of issues, and only a very small portion of them was in development to be fixed right now. They were all meant to be fixed, but the process was slow compared to the supposedly ready system in a week. Part of the reason was that fixing issues from the last sprint took longer than expected, and we didn't have time to even test some functionalities before the acceptance testing started. Thankfully we had plenty of personnel for the testing, but it was looking uncertain, nonetheless.

We received an email today stating that some issues given to developers were not part of the requirements and thus should be noted as improvement suggestions and not issues to be solved. We'll have a meeting about these next Monday, where our side needs to be ready to defend our case and hear the developers.

I spent a lot of time replicating issues others had noted on the Excel. If I was able to replicate the problem, I reported them forward. They had found important errors, and on almost all the issues I tested, I agreed there was something wrong. These were functionalities I had already previously tested but missed these problems. Some I had simply not thought about, and some I lacked the professional expertise to see them. That's why the order was perfect. Our small testing team found most issues and got them fixed before the larger audience came in. Then the larger audience found the finer issues and those that only the experts knew to look for.

3.4.4 Week analysis 4

User acceptance testing is the last phase before the actual launch, and we performed the end-user testing. Besides myself, everyone else from the testing team would use the system after launch, thus making us the end users. Goal of user acceptance testing is to find potential missed issues during functional testing as well as to valuate if the system is on par with the requirements. According to Hambling & al (2015, 51), "The purpose of acceptance testing is to provide the end users with confidence that the system will function according to their expectations." System should

support the business model it's used for, and that's why the end users are the testers now. Developers assist in the testing as receivers for the issues found and giving some training to the end users. Because our project has been made in agile environment, we, the customer have been very involved throughout the development process. That's why our smaller group dedicated for the project did a large part of training by ourselves. Having an agile project also means that we had a good understanding of the system for the most part and didn't just see it for the first time now. After all, we tested most of the functionality during sprint already. Edeki (2015, 22) points out that using agile methodology enables customers to be more involved in the development process. The increased involvement leads to many benefits, including the better understanding of the new system before UAT-period has even started.

Acceptance criteria should be defined between the vendor and customer upon acceptance period (Naik & Tripathy 2008, 451). We also had such a criterion for our period. Essentially, to meet the criterion, our product had to work in a satisfactory manner in the areas that would be used by the customer from the beginning of year.

An acceptance test plan is usually made for acceptance period, as covered in the Testing discussion section. This plan is often developed by the vendor (Naik & Tripathy 2008, 461). In our project, the customer made the acceptance test plan instead, as we had a better picture of the business requirements to be fulfilled. We had the different parts of system in a table, and a spot then would be filled after a person had tested it, with more than one person being able to fill the same spot. This way we could monitor that every spot will be filled, and everything will be tested. With the plan, we also made testing instructions to our test group. Having such testing instructions help people unfamiliar with testing in various ways. One big benefit is increase of standardization. According to Naik & Tripathy (2008, 514), standardization increases the understanding of different terminology as well as decreases different ideologies between people in same organization. In other words, making the instructions should help everyone in understanding the perspective we want them to tackle the testing from. Standardization is compared metaphorically to traffic rules in Figure 7.

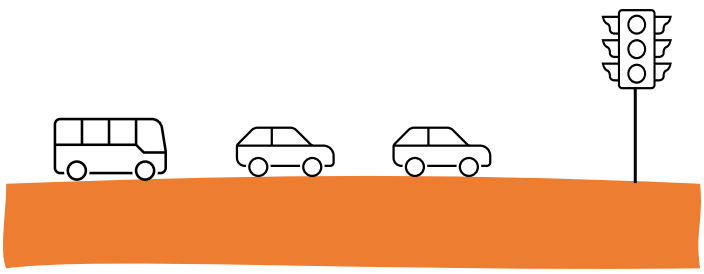

Figure 7. Standardization means everyone follows the same rules, like in traffic.

To keep track of issues, we had the Excel for everyone's use. Because it was used by people that were not usually testers, there was also a lot of unnecessary notes. Suggestions on how the system could be improved isn't usually a part of acceptance testing, and our scope was mostly just actual errors as well, not quality of life improvements. Especially things that are simply different from earlier system but not wrong was not in our scope, but there were still quite a few notes about fixing that type of functionality. Our goal was to make all testers aware of the scope by our own preparation meetings and clearly, we didn't succeed to the extent we had hoped. That meant more work for us as we had to filter the issues more carefully. Liebel & al. (2013, 22) mention that some customers expect to get prioritization for fixing problems that relate to them, as a reward for performing testing. Our smaller team filtered the defects reported by others for this reason as well, among other things. Even if for whatever reason the priority rating was not accurate, it wouldn't reach the developers in that rating, as we would double check the defects first.

We faced some communication problems between us and vendor this week. It turned out, that we had some different opinions on some features and their accuracy towards requirements. Ghule (2014, 547) identifies bad specification of requirements to be a big issue in projects, leading to misunderstandings. In our case, we had failed to express a specific need from the system to the extent that was necessary, giving the impression that less functionality was enough. Additionally, some requirements were discussed in a meeting, without necessarily writing it down, at least not in such detail. This all built a gap between what we'd have needed and what was delivered at some parts. Further, both parties, us customers and the vendor had a different idea of an acceptable result. Resolving discrepancies between the user and vendor is part of UAT (Naik & Tripathy, 463). That's why we'll hold a meeting next week, discussing if our defined requirements justify the need for further development or not. If there's not enough to justify the need, our side we'll need to consider if we are willing to add this needed functionality as extra, and naturally paying for it separately as well.

Large amount of testing has been needed, as there is a lot of different parts inside the scope of the project. Most of it had been tested before UAT-period by our team, but the end users brought additional insight which led to more corrections in the system. As earlier mentioned, some functionality had been delayed, and that functionality hadn't been tested by us before this. When receiving a new part for testing, there is always a lot that don't work exactly right. Because we couldn't do the pretesting for the delayed parts, they now had a large amount of. Our small test team had to use our time for testing that functionality mostly, as otherwise it would have lacked the proper testing altogether. Because we wanted to give support to the bigger testing team by offering meetings every day, our own testing time grew short. We found ourselves not having enough time for testing and couldn't do it as thoroughly we would have preferred. There is still next week, but for now it

feels overwhelming, how much is still to be covered. Of course, this period is mostly to test how the system supports the business model it's intended to. I'm not an end user myself, so my testing isn't as valuable at this point compared to the experts.

Our support meetings to other testers were not led by me, but other members of our smaller tester team. The subject was still within my expertise, so I played the role of support. I answered questions in the chat and added my insight vocally, if necessary. This gave me a different perspective for presenting, as I wasn't in a hurry to write notes while speaking but could delve into them deeper as others presented. This allowed me to have understandable notes, that I could use after the presentation. As the meeting's theme was support for others, I could write up their remarks that popped up regarding the system, and then later forward them to developers. Usually, I find the issues and write them by myself. But when if I try to understand other people's writing, it takes much longer time, especially if I must replicate their issue, which I often have do to make sure it's a real issue. Not taking charge in these meetings, I could find the issues through other people much more efficiently from the background. The benefit of these meetings has been clear for me, and I will try to have them in future projects I'm part of as well. Negativity about them has been my own decreased testing time, but overall, the benefit outweighs the loss because as a team we could find more, and different issues compared to if I was testing by myself.

3.5 Observation Week 5, second week of first UAT-period

Acceptance testing continues, and we will continue to instruct others on their testing experience. As time has now passed, we are also reporting the issues discovered during the period onwards to developers. When time permits, I perform some testing of my own as well.

3.5.1 Monday 14.11.2022

We had the meeting with developers discussing whether the noted errors would be defects or suggestions. It was an important distinction as the other would be included in the initial plan and would have to be fixed and the other would cost us extra and might not be fixed at all. Our side agreed that one issue had been forgotten from the requirements and thus would cost extra. It was about attaching files. There was a general place for attaching files, but we needed specific places for different types of files. But from developer perspective, they had delivered a possibility for attaching files, thus filling our requirements. We decided on discussing it further later, it not being as high a priority as other things.

I didn't do testing by myself today, but only went through the notes others had made. This was quite slow as I had to verify through their forms if there had been something wrong or not. Sometimes they had already sent the forms forward in the system as well, which meant I couldn't edit

them anymore. In such cases, I had to start a new form and fill it the same way they had, so I could replicate the issue. Thankfully this wasn't necessary for all of them, but even the few made my work quite slow. Testing was done by multiple parties now though, so more was found, despite my work having turned into verifying others and submitting the issues to developers.

3.5.2 Tuesday 15.11.2022

Our discussion about the conflicted defects continued today. We came to an agreement to develop some of the discussed functionalities in now during this acceptance period, concluding there being enough time for them. The functionalities were simple enough, and critical. I didn't attend one meeting so I could be able to test as much as possible.

I was able to do a lot of testing today. I continued my reporting work by going through what others had noted, verifying them, and sending them forward. I had plenty of time for it, which resulted in many issues being sent to developers. It was difficult for me to do this kind of work fast, as the notes written by others were not very clear. Some were missing the pinpoint where the issue was found for example. Because of these things, it first took me a while to understand the problem, and then I usually had to verify the issue by mirroring the steps, and only then I'd know if it was worth reporting. This led to a lot of time wasted because process was lengthy, and sometimes didn't even lead to an actual issue. I didn't know a way to make the process more efficient and would have to research the topic by the next acceptance period. Our method did lead us to many discovered defects, it was just slower than I'd have liked, because our time was limited.

3.5.3 Wednesday 16.11.2022

I presented the issues I had gathered yesterday to my team. Most of my day today was used for then writing them down for the developers. As I had first filtered the notes myself, the presenting took less time, and we were able to discuss the right problems. As an example of a defect, there was a lack of feedback when an unsupported file was added, which resulted for the user to not know how to proceed. They didn't know if the file was registered or not. Another more challenging problem was with the relations between two pages. When numbers were added into the previous page, they would not carry over into the next page in some rare cases. The solution was to clear out the data on the previous page in the rare event, making sure there would be no inconsistencies in data. Problem only happened rarely, making the solution sufficient. Even if the user had to apply data twice it was preferable to faulty data. This was a matter of prioritization and control.

I also prepared for tomorrow with the same defect-filtering work I've been doing. We had the helping session as well, where I told what to focus on now. Direction was based on the two days left on UAT-period, choosing the highest priority sections in the system.

3.5.4 Thursday 17.11.2022

This was the last week of acceptance testing and my last day of work for the week. System should meet the acceptance criteria now. There were small errors here and there, but the system was in a sufficient state. Overall, the variations of possible outcomes in system exceeded the amount of people testing, which is why the state of system was sufficient, compared to fully exempt of issues. As discussed on the Introduction section, testing can't clear a system from bugs completely. Certain level of coverage should still be achieved, and it could be argued that our coverage didn't quit reach it, but we'd proceed to production still. The developers will be fixing the issues that have been noted during the acceptance testing for the remaining of this month. It's hard to tell what the situation would be after those fixes, if there would still be a lot of issues or the system would be ready then. It's the area of expertise of project managers, which is why they made the call of proceeding.

My final day of testing was exploratory testing, where I tried to break the system from everywhere I could think of. Thus, I used the method of error guessing. I didn't go to the very limits of the systems capabilities, as it wasn't feasible to expect user going there either. I tried breaking it with actions a user might take. This meant for example choosing a certain option on earlier page, changing data based on that option on the next page, and then changing the earlier option back.

I did one part of the form in a wrong order, which ended up being an issue for the system, because additional data was created because of it. This was a clear error that was then reported. Finding it was a group effort, and not the simplest one. Many testers had reported one table failing to show data. I tried to replicate this error not being able to, concluding it being a temporary error. I told this to the testers, but today one of them told me they had the problem again. I was able to see their form and thus pinpoint the problem to the order of doing things as well as faulty functionality in one button. Because I was able to realize why the issue was happening, I was able to tell the developers more accurately the needed fix, making their job easier and faster to do.

3.5.5 Week Analysis 5

Our system had many different variables, making the possible combinations in different forms endless. To accept the system, we needed to do a lot of work to reach a satisfactory test coverage. According to Hambling & al. (2015, 16), fully exhaustive testing is not possible. The key is to find the balance, minimizing issues that would reach end users. This ideology is supported by what I wrote in the key concepts section; Testing doesn't remove issues entirely, only minimizes them, making the product acceptable to end user. Struggle of balancing time, test coverage and number of repairs is demonstrated in Figure 8.

Figure 8. Accepting the system means balancing time, test coverage and resolving problems.

After an acceptance period it can be harder to argue that a newly discovered issue should be fixed under the same initial cost of development. One tactic used for such issues is to have these fixes be paid by the hours it takes to develop them. This is because everything in the scope of the project should be found by the UAT-period is over. Naik & Tripathy (2008, 450) describe the developed product to be ready for delivery after UAT-period, making it final. Our situation is different from the default because the development got delayed. Delays made our testing period shorter and less effective. It's not as plausible to say that we should have found everything in the period given to us when the period was shorter than initially anticipated. Then again, lacking requirements has slowed the production down at times as well, which has mainly been our fault. In projects like this, there's never just one reason for delays, which is why it's difficult to point fingers to whose fault it is. Like described in Week analysis 4, during UAT-period we had disagreements on if some defects were part of requirements or not. During meetings about the subject, we were able to find solutions that were agreeable to both parties. It was important for us, the customer, to find an agreement, because the wanted functionality was crucial for us.

There are three possible conclusions to come out of UAT-period: Accepting the system, accepting the system after the modifications or repairs are complete, or rejecting the system (Naik & Tripathy 464-465). Before the UAT-period began, we already planned to do most of the repairs after the UAT-period. In our project, the middle conclusion was the natural one. We'd accept the system after the modifications are complete. We do have many defects reported though, and a significant amount is still waiting to be reported. Time for fixing will be until the deadline though, but preferably not during December anymore because of holidays.

My time management is improving. I was able to use some days almost exclusively for testing, and when I led a meeting, because of good preparation it went smoothly. I had done the work I could before the meeting, so then the meeting could be used only for its purpose. In one meeting we

went through the found issues. Because there had been many issues initially, it was crucial that I had first gone through them myself and picked the important ones as well as wrote notes about them. This week I also didn't attend one meeting, where my participation was less valuable compared to using my time for testing. In testing, we must prioritise what to test based on their importance. I used the same principal and prioritised my day usage based on the importance of different tasks. As I had mentioned on last week's report, I had to make better use of my time going forward, and this has been a step to right direction.

3.6 Week 6, After first UAT-period and onwards

This week's reporting is split in half. First two days describe the aftermath of acceptance testing. After them, diary is continued from March. Second acceptance testing period is set to start 20.03. and to cover both UAT-periods, a skip in time was made. Week analysis 6 will describe the main events that have occurred in between November and March, on top of the usual analysis.

3.6.1 Monday 21.11.2022

The acceptance testing period was over. There were a lot of defects that were in development and need to be tested again when they were repaired. I oversaw those returned defects, forwarding them to others as well as testing them as much as possible. As the main part of testing was now over, other people were returning for their original responsibilities. In turn, my responsibility for testing increased. Number of defects to be tested was 12 now, with more quickly to come.

I reported my last defects discovered on UAT-period today as well. On one page a functionality didn't work on some of the forms. One textbox was supposed to carry the data to another page but failed to do so. Because of the failure, later parts didn't work properly either. We had a lot of different kinds of forms, and because the issue was not in all of them, I had to sort out which worked, and which didn't.

I spent some time trying to find the reason why the system saved a wrong kind of file and presented it to the customer. This was a defect discovered by coworker, so I had to understand the issue first. System was supposed to save an approved pdf, but instead showed a png-type picture of a logo from my institution. I was trying to copy the form exactly to replicate the problem. Usually, the first step in fixing an issue is being able to redo the issue. After replicating the process, I was not able to get the same error though. This made me believe that the problem wasn't in the system but rather in the process, somehow. As I wasn't the one who originally made the faulty form, figuring out the problem was more difficult. We shared this issue with the production and together we were finally able to figure out the issue. We execute the approval of forms via email, and in those emails is the signed approval attached. The sender signed the approval and had the logo of the

firm attached to the message. The system was not prepared to have more than one attachment in the email and chose the first one to be used. To fix this, we'd have to either not have to attachment in the message, or have a distinct name for each, so the system could pick the right one. After being able to identify the problem, we had more than one possibility for solution. But the identifying itself was the biggest problem, solution would be much easier.

3.6.2 Tuesday 22.11.2022

There were about 20 of the fixed defects to be retested today, and I was able to check half of them. I'd delegate them to others in my team if the amount became overwhelming, but for now I was handling them all by myself. I knew by now that I had to plan before I'd start working, as there was too much work to do if I was unorganized. I sorted the defects into categories based on their location in the system, which allowed me to tackle multiple ones at the same time, as I could test all defects from same area at once. We had different types of forms, and I was able to cover multiple defects with just one form. If I hadn't done that, I would've wasted too much time making multiple forms, as even making one takes a long time.

The type of testing differed from previous weeks, as now I wasn't looking for errors, but reading the definition of the fix and checking if the system matched it. This type of testing was much more straight forward, as I mainly just tested and confirmed properly working functionality. Doing this was necessary though, as I still found small errors in the fixes.

I tested everything I could and gathered notes along the way on the tests I had questions about. At the end of the day, I had the questions ready, with a browser tab open for every respective question. My notes were well made, and I was able to communicate my problems and questions efficiently, doing one tab at the time.

3.6.3 Wednesday 08.03.2023

We had a new functionality to test today. The idea was to have a way for the handler from our side to communicate with the customer using the system. When the handler would input a message to the system, it would automatically be sent to customer via email. In turn, customers email response would then appear inside the system as well. I confirmed the email system working correctly, all the needed emails were sent. I noticed the communication failing to show in the system though, so I reported it. Developer confirmed that because of a bug, the system failed to display the messages that had been saved. In other words, data was saved, just hidden. After the fix, I tried again. The messages were now displayed in the system correctly, but now the email towards the customer didn't include the message written in the system. It was a good example of why it's important after

repairing an issue to check other areas of system and not just the repaired part, as something else can break: the part that had broken had previously been accepted as correct.

There was an issue where the data from one page didn't carry over to the next. This very issue had been notified previously and fixed too, but now had stopped working again. Reason could be that fixing something else broke this functionality, like the previous case. I presented the issue to my supervisor, which went smoothly as I was already familiar with the problem.

We found an additional spot in the system, where we should have had better requirements. If a specific piece of data had been added to the earlier form, it shouldn't be allowed to add to the later ones. This was another rare occurrence, which is why we had failed to think about it before. This data couldn't be on both forms as that would've meant double the amount registered, which was not allowed. Thankfully, this could mostly be prevented by instructing the end user. For that reason, the issue wasn't top priority as we could work around it. Once again, we had to prioritize. Editing the instructions was easier than adding new functionality, so for time and resource reasons, we chose the easier way.

3.6.4 Thursday 09.03.2023

Because of communication issues, the problems found through testing didn't reach the developers until our shared meeting on the afternoon. This led to me not having a lot to do in the morning, as the things I was waiting hadn't been fixed yet. We communicated through the system by changing the state of stories. When the state was "customers testing", we had to perform testing on it. In turn, when state was "work in progress", it was developers' turn to work on it. The state could be changed by both parties, after we were finished in our respectable duties. There was no email notification when the state changed though, meaning they'd still need to notice it going to the system, and it hadn't happened yet. I will probably start sending emails on important cases going forward, to ensure the most important work gets noticed as fast as possible.

We had a meeting about the uncertain defects that we'd been testing and reporting lately. We mainly focused on defects that were prioritised high in importance. There was discussion on a certain defect, about filtering what was shown on a pdf. Problem was that it was difficult to write rules that would auto write content for pdf based on the type of form it had been. I was able to prioritise on the go, declaring some content in the pdf to be less important, compared to other parts. This way the developers could focus on the important parts of the pdf. Like for all work, the most important functionality should be done first, and in this case as well. This way if the time ran out, the crucial functionality had still been implemented.

3.6.5 Week Analysis 6

This analysis will cover the week after acceptance testing, and two months forward from that to where we are now in the project.

During UAT-period, it's common to decide on how to proceed based on the process made so far. One of these steps can include continuing on testing for an extended time. Naik & Tripathy (2008, 465). In our project, the system had to be published in January, thus making extensions almost impossible. Our structure of UAT-period relied on the programmers working on the required changes after the period, and they wouldn't have had enough time to do so, if we'd extended the testing period. After the programmers would create the repair, we would need to re-confirm the changes made before the system would be published. All these necessary actions prevented us for extending UAT-period.

We couldn't report all the defects we'd found from UAT-period before the period ended, because of the high amount of these defects. Even though we couldn't extend the UAT-period, we had to report all the defects still. The reporting was thus continued after UAT-period, shrinking the time developers would have to fix them. To prevent this in future, we should have had more extensive system testing, as there were still too many issues during UAT-period, that should have been repaired during system testing. Some of the found issues had happened because of added functionality broke previous already accepted functionality. As explained in Testing discussion, such issues can be discovered doing regression testing. Manual regression testing is slow and would benefit from automation. Another larger problem was that one functionality was added too late for us to test it before the UAT-period, so we had to test it alongside UAT. As we hadn't tested that functionality yet, there were large number of issues with it. This took unnecessary time from us and also the developers as they had to fix the problems. Overall, time was the reason our UAT-period wasn't as big of a success as it could've been.

After our coordinated repairing period was over, we decided to proceed to publishing. Thus, the AOT-period was accepted. To reach this, we reprioritized our defects that had not yet been fixed. We couldn't fix everything during the fixing period, so we had to assess on what could be moved to the second part of the development. That way, we could still move the system to production, and we'd still eventually have everything necessary included in the system. Prioritisation was made based on the business requirements, and what part of the system would be used by the end user first. Due the nature of the business, it was clear what functionality user would need at every given time, which allowed us to push development of unused processes for later.

The following two months consisted of working on the defects that had been pushed later, as well as starting the development of new functionality. The development was done in sprints, similarly to first phase. The customer side was part of this development in performing integration and system testing on the new functionality. Idea was again to minimize workload of second UAT-period. We had three people in our group dedicated for the project, until one of us left the institution in March. I had a big responsibility in handling defects and new test cases. If the amount of them was becoming too great, I could delegate them to other members of the institution, which I did a few times. One person from our group was also making a guide for using the system during the two months that passed. This work took a time from their testing, which meant more testing for me.

As time has passed, I've clearly improved on my preparation for presenting as well as on performing the presenting. This has made the meetings I lead go faster and smoother. Saving time was becoming even more important now, as our small group decreased to two members.

3.7 Week 7, week before second UAT-period

Once again UAT-period is lurking. With one UAT-period behind us, we are now making an effort to improve from last time. I perform black box testing this week as well, confirming the functionality being correct. Some of my work is turning towards management type work, as I'm delegating work forward to others as well as planning the beginning UAT-period.

3.7.1 Monday 13.03.2023

Our testing team had been reduced to two people with the other person being in the managing position on top of testing. Another acceptance testing period was to begin next week, so this week would be used for finalizing the functionality and assuring it was working correctly. We were simultaneously receiving fixed defects for retesting as well as testing the stories covering new functionality. Both things combined was too much for two people, which is why we held a meeting discussing on what to prioritise. We agreed on the stories being top priority. This was because in acceptance testing it was intended to confirm that the system fulfilled the requirements of business. To find this out, there shouldn't be any errors in the functionality itself, all the bugs should be fixed earlier. That's why we would test the stories first, so we would have minimal functionality bugs next week. The defects that were returned were important as well, but less than stories, as the defects that were returned to us had already been repaired once, and thus should be in a better state than untested stories. Last time in our acceptance period, we had some functionality that we hadn't had the time to test before the period began. This led to there being much more issues within that functionality, and it took a more significant amount of our time. We were trying to prevent it this time, as our time would be even more valuable next acceptance period because there was only two of us.

The number of stories was sizeable enough that we had to delegate some of them to others from our institution, who'll be the users of the system as well. We gave them a few larger entities for testing, so we could focus on other parts. While this gave us less things to test, there was going to be the additional work of sorting through the notes made by others now. Usually confirmation was needed as well, to see if what was reported was indeed a problem or not. We were slightly worried if the delegating would lead for more trouble than it was worth, but we also didn't have much of a choice.

3.7.2 Tuesday 14.03.2023

Like yesterday, I was delegating stories and testing them today. On one of them, I had trouble finding a button that would be used for unlocking a testable functionality. I had received instructions for finding it, but was unable to, even with the instructions. This led me to believe that there was a fault in the system, and the button was not available at all. In the past there had been cases where some functionality was only unlocked to admin users, and I suspected it being similar to that. It was also possible I simply failed to locate the button for a third time now, but that was unlikely. If so, the system would have been quite unintuitive, which would be a problem in itself. Testing of this story had taken a long time with little process due the inadequate communication. As we communicate in the story itself, it's reliant that both go inside the system and keep track of the stories, as updates in stories don't send notifications to other apps. In this case the tracking was lacking, and it might've been better to use some other form of communication.

On another case I was testing a functionality where a form was filled after a completion of another form. This new form then would be sent for end user to be accepted. The content of form was the main subject of testing, but I happened to notice there being a rare inconsistent error. One of the intended features included the ability for rejecting the form and sending it back to applicant for corrections if the form was unsatisfactory. During testing this feature, I noticed that when form was rejected twice, an additional unwanted message would be sent to the end user. Noticing the error, I checked similar features from different parts of system and found the same problem occurring. I had failed to think about it earlier while testing the other forms, but at least it was found before acceptance period.

3.7.3 Wednesday 15.03.2023

Day was used on working with stories. Some of the testing consisted of simply comparing the requirements to the result. Doing this, a few forms were found to be lacking. Because the forms were built on top of same base, it was possible that later added feature was added to only one of them and it was forgotten to be duplicated for the rest.

Another functionality was forgotten to be unlocked for us, as most features were initially unlocked to the developers only. When testing by users started, developers unlock the features for us only then, so nothing unfinished would be visible for us testers.

Comparing forms against each other, it was discovered a name moved automatically from previous step to the new form. This didn't happen on one form and happened on another. By comparing the two forms, the issue was found. Usually comparing was done between the requirements and the product, but sometimes comparing a feature in system to a feature in another part of system can lead to success as well.

As we had delegated parts of testing to the experts, I also used some of my time answering emails from them as they were asking about the system. As acceptance testing begins, there might be increasing amount of this, and there's a risk it'll eat up all my time. Last time I minimized it by only answering during the shared meetings dedicated for helping others, and I will probably have to do something similar this time as well.

3.7.4 Thursday 16.03.2023

My work responsibilities were shifting towards those of a test manager. My tasks included reviewing the test notes made by the other testers and planning for the upcoming acceptance testing. I also did some testing of my own. When receiving test results, I needed to evaluate their legitimacy and importance. There were a few notes that were more in the field of improvements rather than fixes. There was a suggestion about a button that was found at the very last page of one form. It would've been beneficial to be found at all the pages. I agreed on the suggestion, but as it was not included in the requirements, it would have cost us extra to be made. I sent a message to developers nonetheless, not necessarily expecting a positive response. Some of the notes were clear issues, like value being different between two forms that should have been the same.

As the first acceptance testing period could have been managed better, we tried to improve it now with my supervisor. We started by creating a testing plan, where we listed all the different larger entities that then would be tested. We planned on directing those entities to different testers. We had received feedback that last time it wasn't very clear what each person should test. Now we would hand out different parts of a system for testing to different people directly. This way everything would be tested, and everyone would know their roles better.

We've had problems with the forms being restricted by time previously. Some forms would only open after enough time had passed. We handled it earlier by changing the time to only one day compared to around 30 that it will be after launch. For the acceptance testing even one day was too restrictive though, so I sent an email about the possibility of disabling the restriction altogether

for next two weeks. This would be a rather simple quality of life improvement, but one I didn't realize to ask last time. Because last time it was reduced to one day which was already a big improvement, I didn't think to expect even more. Now with an experience of a single UAT-period behind me, I'd know to expect even better.

3.7.5 Week analysis 7

The second UAT-period is next week, and the main topic of this week was preparation for it to our best capabilities. Our added team member who'd came for support had now returned to their previous work, leaving us. As we are only two people now, managing our resources will be key to succeed in our second UAT-period. Luckily this is the second time, so we can use our learnings from the first period. This time, we will instruct the larger team more precisely, ordering different parts of the system to individuals, instead of vaguely to someone who has time. Our experts asked this from us, as they wanted clearer orders for their own clarity, but there are other benefits as well. Liebel & al. (2013, 22) describe the need of bigger focus in choosing the right customers for testing. With us choosing on who tests what, we will gain the advantage of having everyone testing parts of system they are most knowledgeable at.

In addition to feedback already received from first UAT-period, we will further ask the input of experts on how to proceed with the period. We want to know what went wrong last time and what the larger team would like done differently this time.

Compared to first UAT-period, there isn't a whole new untested part of system going into the period. This should mean that our time can be used for UAT-period's intended purpose, confirming that the system meets its business requirements. The state we're in right now should allow as to go through the system with smaller number of problems compared to last time. Additionally, we will put more effort into the planning than last time.

This week, we started giving stories to be tested by people outside our testing duo. At this point of the project, the people more heavily involved are quite familiar with the system, so we can give them system testing tasks outside UAT-period. They have gained experience and knowledge from first UAT-period and all the meetings that have been held about the system, be it requirements, discussion, or demos about new functionality. Their time is limited, as their main tasks in the institution are unrelated to the project, and the testing is in addition to their other tasks. Naturally the amount of work we have given them reflects that, and it's still mainly our job to handle the testing. It's still useful to be able to share the work, if others don't need a lot of guidance. This delegation was done before UAT. Thus, the tasks given weren't about confirming the business needs, but rather that functionality was working properly and in according to the test cases. Liebel & al. (2013,

22) highlight in their case study that involving internal users in testing can be difficult due them being busy with their other projects.

As this is the second period, the added functionality from first period should already be working, and in theory we only need to test the new parts. Because sometimes when adding new functionality, old ones get broken again, there might be individual parts that don't work again. As covered in Testing discussion section, finding out such problems is done by doing regression testing. If such problems don't exist, there will be less testing overall compared to first UAT-period, as the number of features added has been slightly smaller compared to first UAT-period. Because the amount is smaller, we should be able to get through the period with just us two people and the end-users.

3.8 Week 8, first week of second UAT-period

For the last week of this diary, first week of the second UAT-period is covered. Managing of testing is performed, using knowledge from the first period. Improvements are attempted to be made for the testing experience for our larger test group.

3.8.1 Monday 20.03.2023

Time issue was solved now, by tuning the restriction to a single day with the added ability for user to modify the starting date, so the one day could be turned into zero days. This allowed creating forms back-to-back and gave us testers the much-needed freedom for testing.

We continued with our plan for the starting UAT-period. We'd present the plan tomorrow on Tuesday. We created an Excel including the different entities to be tested, and a Word document including instructions about those entities. The instructions were meant to give insight into how a certain thing was supposed to work in theory, but not in an exact manner, as that's not the point of UAT. The wanted process was described, and the tester needed to be able to use the system without help to fulfil the process.

I asked for feedback about the plan. As one of the main remarks we received from last UAT-period had been the confusion about each person's tasks, we now received feedback that improvement was made in the matter. Our main goal was to get every crucial feature tested, and ordering people to certain tasks instead of giving freedom to choose was thought to help with reaching that objective.

During testing, I found that a certain table failed to update its content upon inputting data. Another textbox failed to sum two salaries together, one sum being from the current form and the other from the previous form. On the same page there was a similar textbox for adding two amounts of

hours together from different forms. It was basically the same task as the other textbox, but this one was working correctly whereas the other one didn’t. That must’ve meant, that the developers had forgotten to copy the feature to the other textbox. Repairing the issue should be easy, as the command should be able to be duplicated from the working textbox.

I also spent time writing instructions for the testers on how to proceed with some tests. It was a part that testers had difficulties getting past from in the testing environment. As mentioned, there was now a way to bypass the time restrictions of forms. This was a new feature for everyone, and some found the method confusing. That’s why I created instructions on how to manually get past the restrictions. Instructions weren't about how a functionality worked, but rather about how to access the right functionality to be tested.

3.8.2 Tuesday 21.03.2023

We presented our new UAT plan to the team today with my lead. I told about the Excel where people would find their assigned tasks and the Word document where they could search information on the tasks. We also had a document where everyone could report found issues, and our smaller group could then have them all at the same place. I planned for the presentation by writing a few notes and having every document ready. It was my first in person presentation, and it went very well. Group found our plan clear, and they were pleased to have clear understanding what every individual would be doing in this period. While presenting we had some suggestions as well, which I wrote down for later.

My own testing was continuing from where I left off yesterday. I discovered that same issues found yesterday occurred in a wider range of possible forms. I didn’t find any new problems. I also reported a defect on a different functionality, where a button was not visible to the user. Developers instructed me with the conditions that would be required for the button to become visible, but I couldn’t make it visible after filling those conditions. My guess was that there was some kind of additional condition, like the button only being visible to admin users or something. Whatever the case, it was a clear bug that prevented me from proceeding with testing, as I would have needed to press the hidden button to proceed.

I also tested a small fix needed to the published system, where one of the automated email confirmations had been disabled by accident, so the end users didn’t see it for a while. Developers added it back to test environment for me to test it. This was an issue happening for real users, so it was more urgent. They were handled through a different outlet and took priority for us as well. The fix wasn’t pushed to production yet today but would probably be tomorrow.

I would've liked to get more testing done today, but preparing and presenting the plan for others took a long time. Testing additional bugs found in published system ate time away as well, leaving me with little time for my usual tasks. If the objective of the UAT plan is successful and leads to people testing effectively, time used for making the plan will be well worth it though.

3.8.3 Wednesday 22.03.2023

In the morning, we assigned the work to our testing team. Assigning was based on the respective expertise of every tester. This way the goal of UAT, consideration if the business needs of system were met would have the highest chance for success. After the assigning, the plan for UAT period was complete, and I sent an email including all the details about the period. I received positive feedback about it by the manager of our group project, them saying the plan was well done.

We also started hosting a daily support meeting from today on, where we discussed together issues found and any questions testers would have. The meeting was helpful in discovering if an individual had an issue that others had as well, or if only they had it. This helped in figuring out the scope of the problem. From this meeting we also received positive feedback from the attendants, as they found that meeting gave them clarity for testing.

Testing I performed myself was mainly exploratory testing. I created multiple forms that read data from the previous form, work I've done other days as well. Forms receiving data from the previous form and further changing it in next one wasn't simple, and susceptible to bugs. That's why it needed to be tested from many angles. This time I found that the previous repairs had made changing pages impossible at one point in the form. There was a new error message popping up when pressing the button "next page". As described in the Testing discussion, regression testing is performed to find these kinds of problems.

3.8.4 Thursday 23.03.2023

Now, as everyone had been testing for a week, we looked at the issues that had been noted so far. We found ourselves spending more time than intended doing this, as the document hadn't been filled specific enough at times. We had to do a lot of mystery solving to understand what people were trying to say in their notes. The document failed to ask a specific page in the form where the issue was found. It was implied but not required, so some people only told the type of document. Naturally, this made finding the location of the problem slower. The task of reporting the issues forward ended up being too slow for us, so we decided to host a meeting next Monday where we could go through the written issues together and instruct the right way to report issues. In hindsight, we should have done this more thoroughly to begin with, but we didn't realize the document being unclear to people less familiar with testing.

I was responsible for being available for helping testers this week. I had a few calls today, where I showed how to access certain spots in the system. We had prepared instructions for the UAT-period which has allowed testers to figure out many of their problems themselves, but naturally there was going to be some instances where more help was required.

3.8.5 Week Analysis 8

Even with better planning and clearer instructions to testers, I found myself wanting more time for my own tests. It's good to recognize that there have been improvements in the period too though, many problems from the first UAT-period have been avoided so far. During first UAT-period we had been asked to give a clearer instruction for the second period. We had achieved this by directing different parts of the system to certain testers. I learned that more thorough instructions for fulfilling the document where everyone reported issues was needed as well. As mentioned in Week analysis 4, standardization increases the understanding of requirements and best practices for the employees. Instructions in our document surely helped in the understanding, but I realize we needed to make the instructions even clearer. For future, I now understand the benefit of standardization very well, as now people's different documenting methods had slowed our work down notably. By improving the instructions even more, I would surely gain more time, which I could then direct towards testing.

My main testing technique used this week was performing testing based on my experience as a tester. As described in the Testing discussion section, this can be called error guessing. With error guessing, tester gives additional attention to sections of the system with a higher possibility of issues. During the project, one of such sections has been data that moves between forms.

Once again, by performing regression testing, I discovered places in the system that had been broken after adding new functionality, or after something else had been repaired. Lonetti & Marchetti (2018, 6) highlight a trade-off in regression testing, emphasizing the connection between the frequency of performing regression tests and resources required for it. New changes are made regularly, and doing regression testing after each change would cost an impossible amount of resources and time. Still, every week some issue related to this topic is discovered, proving the regression testing to be very essential.

My skills in presenting and instructing in meetings have gradually improved. Now, in the final week covered in the diary, I held a full meeting in person with success. My preparation for these meetings has allowed for them to go smoother, and more efficiently. Additionally, I've been helping testers most days in their issues related to testing. This has been working well also, and most we've found a solution during the meeting most of the time.

4 Discussion

Diary covered a longer-than-usual time-period because I took a break in the middle. This allowed for a coverage of two UAT-periods, as well as a possibility of comparison between the two periods. Personal growth during the time period was also more apparent, because when more time passes, it becomes easier to see change. During the 8-week period using different testing techniques were discussed, difficulties were described, and successes were reported. Project proceeded onwards after the covered diary period and my employment. At the end of the employment, first part of the system had been published, and the second part was close to being published as well. First published version of the system had received positive feedback from the end users.

During both UAT-periods something unexpected happened that took time away from our testing. During the first UAT-period, we had disagreements with our vendor on some of the functionality and if it was part of the requirements or not. Additionally, in our shared document used for reporting issues, there were issues that weren't in the scope of our UAT-period, like suggestions for improvement. We also had untested functionality during UAT-period, which meant the functionality required added care to find all the issues in it. Trying to be wiser the second period, we tried to prevent these same problems occurring. In some parts we succeeded well. During the second period we didn't have a similar untested functionality going in the period compared to first time. This enabled the testing to mostly stay on the correct frames for UAT-period. Related, we put more emphasis on clarifying the nature of defects we were looking for from our testers. We also made the general instructions for the period with personal orders for each tester, further clearing the process. This was a clear improvement, and we received feedback that testers had a better understanding on our expectations for them. Second period wasn't perfect still, and we found that we should've given even clearer instructions for the reporting process. We had a repetition of a problem from first period in this matter – our time was unnecessarily spent on trying to understand some of the reports made by the testers. As explained in the Week Analysis 8, having a clear standardization on this matter would've given a better result.

Another area where we improved for second period, because we learned from last period's shortcomings was time restrictions. As the system restricted some forms to be filled until a longer time period had passed, our testing capacities had been less extensive than what we'd have preferred. First period, we tackled the issue by decreasing this limit to one day. This was a clear improvement and made the testing possible. Still, one day was found to be too impractical, and now for the second period we got a way to bypass the restriction completely. It was a good lesson for me to keep mindful of possibilities even after changes for the better have been made. Sometimes, there's

room for even more improvement. In general, giving additional attention to features with time restrictions is a good idea to minimize problems related to them.

Issues around the topic of regression testing were frequent. It became apparent that I needed to gain knowledge in the field of automation, to better perform regression testing in the future. In many of the weeks covered, a functionality once accepted had been broken again, which was discovered with regression testing. Thus, performing this type of testing was essential and doing it manually was time-consuming. I would need to learn to use automation to ensure high level of product quality throughout the whole project and to keep unseen breakings from added functionality at minimum. Even though for this project automation wasn't possible to be used, its potential benefits stood out at some stages of development.

My presentation skills and preparation needed for it have improved throughout the project. While I was able to hold a meeting from the beginning, those meetings held less value for two reasons. Because I didn't prepare as well for the session, some time was wasted in remembering my previous thought process, and in the worst case, some topics were lost. Additionally, because I made notes during the meetings with one or two words instead of coherent sentences, I might've later forgotten what I had written about. Or at least, it took me a longer time to analyse my notes compared to now. This type of work can feel redundant, but I noticed by keeping a diary that there is value in doing it with more care. Related, I discovered the value of journaling. If I hadn't kept diary for the first UAT-period, I most likely would have remembered fewer details from it. Thus, there would've been a higher chance of repeating the same mistakes in the second UAT-period.

A more unique approach to Agile development was used for the project. Having a separate team of testers on the customer side for the full development lifecycle ensured the project moving towards the right direction throughout the whole timeline. Even with this constant reassuring and steering, we had a significant number of topics to adjust during the UAT-periods, and delays in various parts of development. It makes me wonder, if a project doesn't follow similar methodologies, how do they reach a desirable outcome? From the perspective of precision in the systems features and their correlation to requirements, having a dedicated test group from customer's side seemed beneficial. During various parts of diary, verifying was done to ensure that the functionality acted in acceptable manner. This verifying was frequent enough to become overwhelming if left completely to UAT-period, compared to dividing more evenly throughout the full development lifecycle.

In part, level of competencies related to competency requirements of the job have risen to the next level. In the beginning of the diary, my competencies were in the level of skilled performer. During the covered time, skills falling under a description of an experienced expert have been demonstrated. Having worked in creating a UAT-test plan as well as guiding others in testing, I've partially

gained experience to be considered an experienced expert in the frames of the requirements for this job. To become a skilled performer in the frames of a test engineer I will need to increase my knowledge in the areas of automation and creating test cases. That will be my next professional goal.

As automation has been identified to be an important skill to develop to make testing a career, next steps for me revolve around studying the subject more. I will continue studying in university, proceeding into master's degree. Courses in testing including automation will be taken. Knowledge in robot framework is often singled out in job listing descriptions, so it is given priority in studies.

Sources

Edeki, C. 2015. AGILE SOFTWARE DEVELOPMENT METHODOLOGY. European Journal of Research in Medical Sciences Vol. 3 No. 1, 2015. United Kingdom. pp. 22-27.

Ghule, S. 2014. Risk Analysis and Mitigation Plan in Software Development. International Journal of Engineering, Sciences and Research Technology, 3(8). pp. 546-548. URL: https://www.ijesrt.com/Old_IJESRT/August-2014.html Accessed: 1 January 2024.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., Williams, P. 2015. Software testing - An ISTQB-BCS Certified Tester Foundation guide 3rd edition. BCS Learning & Development Limited. United Kingdom. E-Book. Accessed: 10 November 2022.

International Software Testing Qualifications Board (ISTQB). 2014. Glossary: Standard Glossary of Terms used in Software Testing. V.3.7 URL: https://glossary.istqb.org/en_US/search. Accessed: 24 September 2023.

Jamil, M. A., Arif, M., Abubakar, N. S. A & Ahmad, A. 2016. Software Testing Techniques: A Literature Review. 2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M), Jakarta, pp. 177-182.

Liebel, G., Alegroth, E., Feldt, R. State-of-Practice in GUI-based System and Acceptance Testing: An Industrial Multiple-Case Study. 2013 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Santander, pp.17-24.

Lonetti, F., & Marchetti, E. 2018. Emerging Software Testing Technologies. In Memon, A. M. (eds.). Advances in Computers Volume 108, College Park, pp. 91-143.

Naik, K & Tripathy, P. 2008. Software Testing and Quality Assurance: Theory and Practice. John Wiley & Sons. New Jersey. E-book. Accessed: 4 December 2023.

Orellana, G., Laghari, G., Murgia, A., & Demeyer, S. 2017. On the Differences between Unit and Integration Testing in the TravisTorrent Dataset. MSR '17: Proceedings of the 14th International Conference on Mining Software Repositories, Buenos Aires, pp. 451–454.

Appendices