Test Plan

Company 1 - TDDC88

Contents

Version	Author	Updates	Reviewed by	Date
0.1	E. Sköld	Initial version	A. Telenius	2021-09-13
0.2	E. Parö	Test Levels and more	D. Ma	2021-09-24
0.3	A. Telenius	Fill in last headers	D. Ma	2021-09-24
1.0	A. Telenius	Rewrite every section in chapters 2.3-4.2	E.Parö	2021-11-07
1.1	O. Löfgren	Rewrite chapters 1 - 1.3	E.Parö	2021-11-09
1.2	E. Parö	Rewrite chapter 2.1	A. Telenius	2021-11-09

1 Introduction

In this document an explanation about our test strategies, process, workflow and methodologies used in the project is given. The test plan will describe the structure of our testing and how they are incorporated in the process of developing the product. Every type of test that we will perform is going to be further explained into detail. The reason why we will perform certain tests at which stage in the project will also be explained. This is going to be used for all the testers to work in the same way and to have a standard within the company, this test plan will be read by new testers for learning how to work with the testing. Therefore there will be further explanation of how to handle bugs found during testing, what tools that are used and when a test is considered complete.

We follow the V-model which is a model that has a testing phase parallel to the development phases. This model is also called Verification and Validation model, it is an extension of the waterfall model as it is also a sequential model. There is a testing phase parallel to each development stage. This means that it includes testing for the requirements, system design, module testing and the implementation. The advantage of using this model is that it is simple and easy to follow and when the requirements are well defined, it also works well for smaller projects like this.

The following test plan is derived from the test plan template created by Thomas Hamilton for Guru99. A link to the template can be found here.

1.1 Scope

The scope of the test plan is defined below. This includes what will be tested (In Scope) and the parts that will not be tested (Out of Scope).

1.1.1 In Scope

In accordance with the V-model, testing is to be conducted in unit, module, system and acceptance level testing.

The customer wants us to focus on the usability of the system and how to visualise data in a satisfying way. Therefore the testing will focus on that and measure if the developed application achieves requested levels. The testing will also focus on the functionality of the system so that the intended units and integration between units works as expected. The testing in this project will include Unit testing which tests the individual units, module testing which tests the integration between the different units, system testing which tests the integration between different modules and also acceptance testing that verifies the requirements. All these tests and how we are going to perform them in the project is going to be explained in the test plan. The main focus of the testing is to ensure that the requirements are met in the finished product.

1.1.2 Out of Scope

Due to the limited implementation levels of the project the test plan will diverge from the V-model by not conducting tests at the final level, maintenance.

The customer does not want us to focus on how to store data and database management and therefore such tests will not be conducted. Because the storing of data is not important for this project, security will neither be tested during this project. The testing of the database could otherwise have been done by doing Structural Database Testing which validates the elements that are inside the database that are stored and cannot be accessed or changed by the end users.

1.2 Roles and Responsibilities

Testers
In the first part of the project (iteration 1 and 2) every tester is part of a cross-functional

team (CFT) and is responsible for performing tests for their CFT individually related to the unit, module and system testing. When merge requests are created from a teams branch to the develop branch the tester that is in that team is responsible for the testing of that merge request.

From interaction 3 and forward there will be a specific team focused on testing, this means that the testers will work more together and are not responsible for specific merge requests. Instead the testers will take on responsibility for a merge request by assigning themselves for testing that merge request in Gitlab.

When a bug is found the tester is responsible for reporting it to the developer. This is done through not accepting the merge request and writing comments explaining what the bug is and the suspected underlying issue. If there is a bigger problem that will take longer than two hours to fix a new git issue will be made for that specific problem.

Test leader
The test leader is responsible for the acceptance testing and will coordinate this with the help of all the testers. The test leader will also be the contact person for non CFT-related inquiries for testing procedures. The test leader will also be responsible for setting up meetings with the customer for performing the acceptance testing.

2 Test Methodology

2.1 Overview

The testing is mainly performed by the test team and the testing cross functional team. The test team will have all the responsibility of testing from unit testing to acceptance testing. In the testing cross functional team, we have a analyst and software quality manager which will help validation and verification in higher level testing. The testing is done iterative which mean that all levels of testing is done in each sprint, starting with unit-level testing and finishing with testing the present application with the customer to validate our requirements. This will help us to continuously improve both our application and our testing through each iteration.

2.2 Test Levels

In this chapter the different test levels are presented and also which test and tools that will be used in the different levels. The different testing levels are unit testing, module testing, system testing and acceptance testing.

2.2.1 Unit testing

Unit testing will occur when a developer is doing a merge request to their specific cross functional team branch. Together with the unit testing, a peer review of the code is done by a developer to ensure that the functionality is correct and that the code is following company standards. In addition a pipeline with automated test is used according to continuous delivery. These test should focus on black box testing and will test the most basic functionality of the application for every merge request. The test are written in angular and are executed with karma. The main unit testing will focus on GUI testing through exploratory testing where the tester will try the new unit such as clicking a new button and seeing if the response is right. When performing exploratory test it is important to try as many testing sequence as possible because the user can cause all these sequences.

From iteration 3 and forward the new cross-functional testing team will be responsible for the unit testing. The customer wants the project to have a major focus on user experience and usability of the system. Therefore, the testing will not focus on black box testing aside from the automated test.

The work flow for testing new merge requests:
The developer posts a merge request in gitlab. The automated tests in the pipeline is performed. A company member performs a peer review according to the set up rules. A tester performs a exploratory test on the new feature. If a small fault is found then a comment is posted on the merge request and the developer how posted the merge is responsible for fixing the comment before the merge can be done. If a major fault is found, a fault that takes approximated over 2 hours then a new issue is posted about the bug and the developer is assigned to the issue.

2.2.2 Module Testing

The module testing is the test of the integration between several units which verifies the module design. The testing will follow the top-down approach were we use Selenium IDE test cases. This will be used to test so that the different units will fit together. Test cases will be set up to test the whole application. This is done to make sure that new functionality does not affect already existing.

The test team is responsible for running the Selenium tests. These tests are done simultaneously as the unit test, when a merge request is posed in gitlab. Selenium is used to help reduce the workload for the tester. In a event-driven environment the user can cause events in different order which mean that there are a lot of different test cases. Therefore, the test team sets up test cases in Selenium IDE that can be done for every merge request. The Selenium test is also integrated in the pipeline to maximize the automated nature of the testing.

2.2.3 System testing

The test team is responsible of the system testing which will verify the system's architecture and high-level design. This chapter is dividend in to two sections, function testing and performance testing. This testing level verifies the requirements.

Function testing
To test functional requirements, functional testing is done. This is done to make sure that the application meets the requirements that are made by the analyst team. Because the developed application should focus on the usability and visualising data then this test will also be done exploratory. The exploratory function testing will be done by going through the requirement list and the analyst will either mark the requirement as done or not. We are doing exploratory testing for this phase because the application has a major focus on the front-end. This is done at the end of each sprint to understand which requirements that are done and which are not. This will enhance the traceability between the requirements and the developed application.

Performance testing
To test non-functional requirements, performance testing is done. This test is done to reduce performance bottlenecks. The non-functional requirements will be tested in the same way as the functional requirement if that is possible.

2.2.4 Acceptance testing

To validate the requirements the test team will perform acceptance testing. These test are performed with the customer to understand their true needs and wants.

The acceptance test will be performed with the concurrent think aloud method. To get a quantitative result and measure the usability of the application a SUS-test will also be part of the acceptance test. To conclude that our developed application has a sufficient level of usability the SUS-test must achieve a score of at least X(not yet decided) for the final product. The concurrent think aloud questions and SUS-test will be added as a appendix.

The test will be conducted with the customer after each iteration to gather their response. Because the test is done after each iteration this enables the developer to get feedback of the not-yet-complete application by the customer. This will hopefully make the application tailored for the customer. The developed application will be used by people with different professions and technical experience. Therefore, it is important to conduct these acceptance test with different level of technical experience and profession to get a complete picture of the true needs of the customer. Unfortunately can the customer not provide people with different profession due to specific profession does not have the time to spare. Therefore, some perspectives can be lost nonetheless the doctors, nurses and assistant nurses works close together and can give some insight on what other profession need.

Dates for test with customer:
After iteration 1
Meeting with customer, IT-staff and nurses participated: 11/10 - 21
After iteration 2
Acceptance test 1 with nurse: 11/11 - 21
Acceptance test 2: 12/11 - 21
Acceptance test 3: 15/11 - 21
After iteration 3
Purpose date for acceptance test: 22/11 - 21
After iteration 4, Final product
Purposed date for acceptance testing: 3/12

After iteration 1 the application was working but there were to few features to perform worthwhile CTA and SUS testing. Therefore, after iteration 1, the application was shown to the customer along with the prototype from the tollgate meeting. The meetings purpose was to give the customer an opportunity elaborate on their specific needs for different parts of the website, both functional and design related. From iteration 2 and forward the customer acceptance testing will all follow the same structure. The customer will be given a certain amount of task that they will perform on the application while performing CTA. Afterwards, they will do a SUS-test. The task that the customer will be given can differ between the iteration because more feature will be developed over time thus, we want to include these feature in the test.

Each test with the customer will result in a acceptance test report. The different test reports for each test occasion will be summarized to a single test report that will be distributed to the whole company. This test report will be the foundation of changes in requirements and UX-design for the subsequent sprint. The test report after iteration 4 will show if the usability requirements.

2.3 Bug Triage

Bugs found during the testing procedure must be solved or have the underlying code generating the bug removed. This is due to all testing (except acceptance testing) taking place when a merge request is created. If a bug is found, the merge request is not approved by the tester. The merge request must then be edited to solve the bug or it is not to be merged at all. Bugs found outside of the testing procedure (e.g by non-testers) shall be reported to the testing team which will investigate the bug, create a Gitlab issue explaining what the bug is, steps to reproduce it, suspected underlying issue and deadline for when it must be solved. This is usually set to the next iteration deadline as known bugs shall not be present during acceptance testing with the customer. The Gitlab issue is assigned to the author of the code.

2.4 Suspension Criteria and Resumption Requirements

Due to testers only working on testing and the small, limited number of functions in the application the suspension criteria is only a complete system failure rendering the application non-functional. Testers only working on testing means that continued runs after a program is not working correctly is reasonable in our time schedule. This is supported by the fact of a

small, limited number of functions making each test short. This means that the only resumption criteria is when a non-functional program is made functional and thereby resuming testing.

2.4.1 Exam period, 19/10-21 → 30/10-21

During the exam period neither developers nor testers will have time to perform testing or programming because they will study for their exams. Because all company participants have this period at the same time this will not create a bottleneck, only a temporary stop in the continuous delivery.

2.5 Test Completeness

A test is considered complete when it passes the Gitlab pipeline, the selenium IDE-tests and exploratory testing for each merge. Not passing one of these tests means the merge request will not be approved.

3 Test deliverables

This chapter regards deliverables created for customer, CEO, company and/or testers.
In the beginning of each iteration we intend to have a meeting with the customer where they will do a System Usability Scale (SUS) test with the concurrent think alound (CTA) protocol. Every iteration is roughly two weeks so we planned to do these test four times with the customer during the project. This is to test the new features that have been developed during the iteration that has been so that the customer can give feedback to what they like and don't like. The feedback from the customer will then be delivered to the developers and UX-designers so that they can continue to improve the features and modules. This feedback is summarized by the testers, but the original meetings are recorded if possible and uploaded to enable the possibility to go back and analyze a specific meeting. All results from the meetings will be saved to the "Testing" Teams channel.

The first customer meeting did not feature a SUS-test with a CTA protocol as the application was deemed too bare-bones to result in any meaningful feedback from a customer. Instead, a customer meeting was conducted where design ideas from the Figma protype and the application were evaluated on a detail level and written down. This interview was done with nurses and IT-staff from the customer. The notes from the meeting were then passed to the UX-team for evaluation.

Unit and integration testing is done at each merge request and bugs found will be commented in code and as a comment in the merge request text fields. The creator as assignees of the merge request are then notified by the automated mail sent to their Gitlab accounts of the discovered bug.

The test plan is created to standardize and inform of the testing procedure. It is written by the the quality coordinator (namely **??**) and the testers. It is published both in the Gitlab repository as well as the Teams output folder for CEO and company to be informed about how testing is conducted. When a testing procedure is changed or created, it is written down into the test plan by testers so no procedure goes undocumented.

4 Rescource and Enviroments

This chapter regards testing tools and versions used by each tester to ensure the same prerequisites for each test done by the testers.

4.1 Testing Tools

The requirement tracking tool is verbal communication between testers and analysts where either a test is shown proving that a requirement is reflected in the application or by evaluating the application through existing functionality or properties the application possess. The requirement is then marked as done in the analysts document of requirements.

One of the two bug tracking tools is the merge request comment fields where information about what the bug is, steps to reproduce it and suspected underlying issue is relayed to the authors of the code and creator of merge request. The other bug tracking tool is git issues containing the same information as the first bug tracking tool and is used when bugs are found outside the testing procedure.

The automation tools are firstly the automated test done in Angular and implemented in the Gitlab pipeline which is conducted for each merge request. These tests are primarily used to ensure that the application is still having its basic functionality and is not the primary focus of the testing procedure. This is done due to the heavy focus on front-end testing as the customer application is heavily focused on front-end development. The second automated tool is a list of Selenium IDE tests and are run by the testers after they pull the merge request to their local machine.

4.2 Test Environment

- Angular 12.2.12
- Node 16.13.0
- Selenium 3.17.0

5 Terms/Acronyms

A list of terms or acronyms used in the test plan.

- CFT - Cross functional team
- SUS - System usability scale
- CTA - Concurrent think aloud

References