Software Quality Assurance Plan (v. 1.4)

Company 1 - TDDC88

Contents

Version	Author	Updates	Reviewed by	Date
0.1	Erik Sköld	Initial version	-	2021-09-14
1.0	Erik Sköld	First plans for documents, development, testing and quality	Daniel Ma	2021-09-23
1.1	Erik Sköld	Updated guidelines for development regarding workflow, software reviews and version handling	Jacob Karlén	2021-10-13
1.2	Erik Sköld	Updated scope, moved document guidelines to Project Plan, comment missing parts in Testing, rewrite section 5 to describe Software Quality Assesments	Daniel Ma	2021-11-05
1.3	Erik Sköld	Reformulate document monitoring, add references to other artifacts, update development workflow, develop testing section, change software metrics	Daniel Ma, Oscar Löfgren	2021-11-24
1.4	Erik Sköld	Recreated estimated resources and moved to section 1, developed the section about testing quality assurance and metrics	Daniel Ma, Axel Telenius	2021-12-01

1 Purpose

The purpose of this Software Quality Assurance Plan is to establish the processes and responsibilities required to implement effective quality assurance functions for the project.

The Software Quality Assurance Plan will provide the framework necessary to ensure a consistent approach to software quality assurance throughout the project life cycle. It defines the approach that will be used by members of the company to monitor and assess software development processes to provide objective insight into the quality of the software and the product. The systematic monitoring of the product and processes will be evaluated to ensure they meet requirements.

1.1 Scope

This plan covers Software Quality Assurance activities of the project and will be updated regularly.

1.2 Project description

The project's goal is to develop a tool that visualizes and compiles information from patients at Region Östergötland hospital's emergency departments. The information comes from the various computer systems that Region Östergötland has available today. The tool should compile information in an easy and clear way. The compiled information must be presented but does not have to include input solutions.

The product will be developed during four iterations. The project will continue until December 16, 2021, when version 1.0 will be presented.

1.3 Estimated resources

Task	Estimated hours
Education	12
Developing Quality Processes	25
Documenting Quality Processes	25
Monitor Documents	14
Testing Quality Assurance	8
Software Quality Assessments	8
Supervisor Meetings	8

The estimated resources presented in the table above applies for the Quality Coordinator of the project. Education covers both learning about quality assurance and software metrics as well as internal workshops held within the company. Developing Quality Processes and Documenting Quality Processes are the two biggest posts since the company starts from scratch. Work with these tasks is mainly about developing guidelines for the company and documenting them in the Software Quality Assurance Plan and on the company's GitLab. These three posts are the main focus in the first half of the project. More information about Monitoring Documents can be found under subsection 2.2, for Testing Quality Assurance see section 4 and Software Quality Assessments see section 5.

2 Documents

This section lists the regulatory documents used in the project and the plan for Quality Assurance related to documents.

2.1 Regulatory documents

The Quality Assurance of Documents apply to the six regulatory documents listed below. They are available in the Output documents folder on MS Teams.

Document	Responsible
Architecture Notebook	Jacob Karlén
Customer Requirements Specification	Sofie Andersson
Education Plan	Axel Nilsson & Daniel Ma
Project Plan	Somaye Gharedaghi & Axel Nilsson
Software Quality Assurance Plan	Erik Sköld
Test Plan	Axel Telenius

2.2 Plan to monitor documents

The document guidelines can be found in section 5 of the Project Plan. The guidelines cover the process of updating and reviewing the content in the regulatory documents. This procedure is done as an internal inspection before each new published version of a regulatory document to ensure quality. Beside these guidelines, the Quality Coordinator shall monitor the regulatory documents and make sure their structure follow the company guidelines. This will be done every iteration.

3 Development

This section is intended to address guidelines for the development and quality assurance of code that will result in the end product. The project is set up with TypeScript. The client (Angular) and the server (Node.js) run in seperate Docker containers and are orchestrated with Docker-compose. For a more detailed description, please see the Architecture Notebook.

3.1 Code conventions

As the project is set up with TypeScript (TS), Google's TypeScript Style Guide must be followed for all TS code. This is verified during the software reviews before each merge, further described in section 3.3.

Names for functions and variables must be written with camelCase and be self-explanatory. Names generated by Angular shall not be changed.

3.2 Workflow

The project uses a form of feature branch workflow. Feature branches are merged with a development branch, never directly to the main branch. Once the code has passed testing on the develop branch, it can be merged into the main branch by an authorized manager.

Iteration 1 & 2

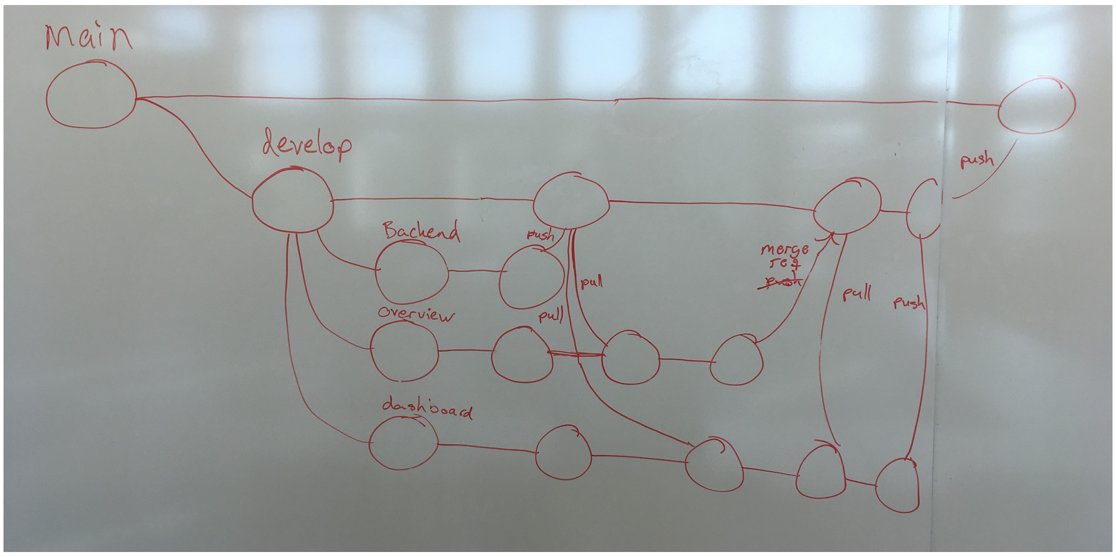

Figure 1: Sketch of workflow used on GitLab for iteration 1 and 2.

There are five long-lived branches:

- *main* - Is updated each iteration, code on this branch must be able to be shown to the customer. Members are only allowed to pull code from develop to main.
- *develop* - The intermediate branch where code from backend, overview and dashboard should be merged to. Code on this branch must be ready to be tested (integration, regression and acceptance testing). Here it is fine to have some bugs.
- *backend* - CFT1
- *overview* - CFT2
- *dashboard* - CFT3

From the three cross-functional team branches, smaller feature branches shall be created. These smaller branches are not supposed to live longer than maximum two days, this is to enable continuous integration. When these smaller branches are merged to their respective CFT branch, peer-reviews must be made (see subsection 3.3 below). All merge requests must be approved according to the set guidelines before it is merged.

Iteration 3 & 4

From iteration 3 the branches *backend*, *overview* and *dashboard* will not be used, they are merged with *develop* by the end of iteration 2. The *main* and *develop* branch remain and new feature branches are created from *develop*. To keep everything connected with requirements, the new feature branch is created from the related issue in the requirement list on GitLab.

3.3 Software reviews

In addition to regular discussion of code between developers, software reviews take place in connection with each merge from feature branches to the development branch. The software review must follow a set protocol in GitLab, the guidelines can be found in the README-file on GitLab. In merge requests, the related requirement must be declared as well as what new features have been added, the expected behavior of these and what to test.

The reviews are done by a project member who has not developed the code themselves within two working days of when the merge request was posted. The review must not exceed one hour. For merges to the main branch, the review must be done by the project architect.

3.4 Bug-tracking system

GitLab is used for bug-tracking. Bugs found during testing will be listed and documented by the test team. Issues for each bug will be created in the issues tab of the company GitLab and connected to the corresponding requirement issue if relevant.

3.5 Version-handling

GitLab is used for version-handling. Work needs to be commited at least daily when working on features but the goal is to commit after each working addition to the code. Commits must be tagged with what kind of commit it is (see the section Semantic Commit Messages in the README-file on GitLab).

3.6 Documentation

The Technical Writer and the involved developers document the code produced at module level. The documentation includes descriptions of which inputs are handled, which functions are used, which inputs are given and how they work together with the rest of the system.

4 Testing

The Quality Coordinator will assure that the test management processes and products are being implemented per the Test Plan. This includes all types of testing of software system components as described in the test plan, specifically during usability testing.

The Quality Coordinator together with the Test Leader will monitor testing efforts to assure that test plans are adhered to and maintained to reflect an accurate progression of the testing activities. It will be assured that tests are conducted using approved test procedures and appropriate test tools, and that test anomalies are identified, documented, addressed, and tracked to closure.

Members of the Test Team will review post-test execution related artifacts including test results and test reports.

4.1 Purpose

Testing will be conducted in order to verify that the product is working as intended. Solutions will be evaluated to reduce the risk of problems and improve the performance of the product. Testing is an important part of the work to ensure that the requirements are complied with and thus assuring quality.

4.2 Test plan

The latest published version of the Test plan is available in the Output folder on MS Teams and on GitLab. The test tools used are different features on GitLab (automatic test pipeline in merge requests, bug tracking with the help of issues and comments), Karma and Selenium. If bugs are found during system and usability testing, they are reported using comments and issues in GitLab. Test results from Concurrent Think Aloud (CTA) and System Usability Scale (SUS) testing are recorded and noted in a document template for each test in the Testing channel in MS Teams. The results are summarized and feedback from the users are forwarded to UX-designers.

4.3 Quality assurance with test cases

To avoid biased testing and a limited coverage, the test plan and test cases are written and developed by designated testers who have not worked with the code that is to be tested. Every test case shall have a clear objective, for example focus on one feature at the time. The test case is broken down into a series of concise steps and when an action is taken, the tester or an automated test should be able to easily measure the success of the action. To ensure meeting expectations from the intended end-user, the usability testing is primarily made with medical personnel.

To evaluate the quality of test cases, the results from SUS testing and the number of comments from users are compared between tests in different iterations. If the SUS score has been improved (a higher score) from one iteration to the next, the usability of the application has improved. If the number of comments from users increase from one iteration to the next, the application probably has more problems and obscure features. Since the purpose of the testing is to find these problems and help improve the application, give both these metrics an indication of the quality of the test cases.

The measurements for these metrics are collected during usability testing in iteration 2, 3 and 4 by testers. The metrics are compiled by the Quality Coordinator in conjunction with the Software Quality Assessments, see subsection 5.1, and documented in a spreadsheet on MS Teams, see subsection 5.3.

5 Software Quality Assessments

This section describes the plan for Software Quality Assessments (excluding testing). The plan includes a schedule of the assessments, their purpose and the metrics to be used.

5.1 Assessment schedule

Date	Iteration to be monitored
2021-11-09	Work until iteration 2
2021-11-29	Iteration 3
2021-12-07	Iteration 4

5.2 Purpose of assessments

The purpose of this procedure is to monitor and measure the product quality. Data is collected to find risks and determine if changes in the product or processes are needed.

5.3 Metrics

For each assessment, the Quality Coordinator will measure and produce the metrics listed below. The results shall be documented in a spreadsheet available to the entire company in MS Teams. A summary of the assessment will be placed in the Output folder on MS Teams. The most significant findings will be brought up at the weekly manager meeting by the Quality Coordinator, where a decision is made whether any further action needs to be taken.

5.3.1 Quality of code

These metrics are noted manually and will indicate the quality of the code with regards to maintainability and understandability.

- Amount of comments per function.

  10 functions chosen randomly from merged code during the the current iteration will be examined. The number of comments from each function are added together and then divided by 10 to get the metric. The metric should have a value above 1 for the code to be able to achieve maintainability and understandability.

- Ratio of commit messages following the company guidelines.

  The commit messages from the current iteration are monitored and determined of they follow the company guidelines. The number of commit messages following the company guidelines is divided by the total number of commit messages monitored to get the metric. To achieve understandability and maintainability, the metric should be as close to 1 as possible.

- Number of files and directories within server and client respectively.

  The number of files in each directory within the server and client directory on the *develop* branch is mapped at the time of the assessment. The measurements will result in a tree structure of directories and files which will be evaluated to see if restructuring is needed to achieve better understandability in the code structure.

5.3.2 GitLab

These metrics are produced from the activities on the company GitLab. The purpose of them is to monitor and evaluate the processes and workflow on GitLab. If the metrics show values worse than the set guidelines, actions will be taken to meet the guidelines.

- Deployment frequency (Planned vs Actual).

  The goal for the company is to deploy code to the *main* branch once per iteration. The metric is determined by looking at the merge history on GitLab.

- WIP amount (amount of "cards" or tasks in progress at the time).

  This metric is measured by looking at the issue boards on GitLab. The number of cards placed on each "doing" boards is added together to get the metric.

- Number of Peer Reviews (reviewed Merge Requests).

  This metric is obtained by checking the number of Merge Requests that have been reviewed in the current iteration.

- Ratio of merges without review.

  To calculate this metric, the Number of Peer Reviews metric is divided by the total number of merges in the iteration. This value is then subtracted from 1 to get the ratio of merges without review.

- Time to merge (time from first commit to merge request sent).

  The time elapsed from the first commit on the relevant branch to when the merge request is sent is measured by looking at commits and timestamps on GitLab. The metric is calculated from the average of the these times from the current iteration.

- Merge request review time.

  The measurements needed for this metric is provided by GitLab in the analytics tab. After setting start and end date, the "mean time to merge" is given by GitLab, which is the same as the average time it takes for a merge request to be approved.

- Number of issues found when reviewing Merge requests.

  The metric is calculated by counting the number issues in GitLab created in the current iteration marked with "bug" and the number of merge requests with issue related comments. This metric is compared to the Number of Peer Reviews metric to determine the ratio of issues per review.

5.3.3 Quality process

The purpose of these metrics is to evaluate the Software Quality Assessments. The values of these metrics will indicate the progress of the process.

- Number of Software Quality Assessments (Planned versus Actual).

  This metric is measured by checking the number of Software Quality Assessments conducted through the number of protocols made. This is compared to the planned number of assessments in the assessment schedule, see subsection 5.1.

- Number of Risks identified as a result of Software Quality Assessments.

  Based on the result of the assessment, risks are identified and brought up to the rest of the company. The metric is the total number of risks identified in the current iteration.