Software Project Management 2021/2022

Flashee

D1.2.2 - Quality Assurance Plan

Authors

- Ângelo Paiva.
- Álvar San Martin.
- Burak Tinam.
- Jan Frank.
- Pedro Henriques.

Reviewer

- 0.86, Ângelo Paiva.
- 0.86, Álvar San Martin.
- 0.86, Burak Tinman.
- 0.86, Jan Frank.
- 0.86, Pedro Henriques.
- 0.86, José Almeida
- 0.86, Rúben Lousada
- 0.88 Burak Tinman

Approver

- 1.0 Burak Tinman

Table of Contents

4. Tests
   4.1. Unit Testing
   4.2. Integration Testing
   4.3. Acceptance Testing
5. Risk Management
   5.1. Risk identification and assessment
   5.2. Risk monitoring and control
6. Coding Standards
7. Quality metrics

1. Introduction

To fulfill our wishes, Flashee needs to be very user friendly, simple to use, quick to master, intuitive BUT, despite all of this, still be featureful. To accomplish this, we intend to give a special focus to UX to ensure that we deliver the experience we envisioned.

This line of thought lead us to make use of flashcards which are a common way to learn things quickly and easily. We will make a digital implementation according to our goals.

The quality assessment report can be found here.

2. Quality Goals

The most important external quality attributes identified for this project are the following:

1. Since our main goal with this project is to make a user-friendly application, I think it's safe to say one of our most important qualities will be the **usability** of our web application.

Regarding internal quality, the following attributes have been identified as the most important:

2. Our project has **high portability** since it will be a **web application** which will not be a problem for any platform.
3. To guarantee usability the project must run **smoothly** in every device, **performance** must be optimized.

3. Reviews

The team and the client examine the project documents and source code and give feedback about it. The development team will do a more technical review and the client/stakeholders a higher level one about the vision and scope.

The review records can be found here.

3.1. Inspections

The goal of the inspection is to reach a consensus about a work product and approve it for use in the project.

- Inspections are moderated meetings in which reviewers list all issues and defects they have found in the document and log them so that they can be addressed by the author.
- Their goal is to repair all defects so everyone on the inspection team can approve the work product.
    - Often, inspected work products include software requirements specifications and test plans.
- Process of inspection:
    - A Product is selected for review and a team is gathered for an inspection meeting to review the work product.
    - A moderator is chosen to moderate the meeting.
    - Each inspector prepares for the meeting by reading the work product and noting

each defect.
    - In an inspection, a defect is any part of the work product that will keep an inspector from approving it.
    - Discussion is focused on each defect and coming up with a specific resolution.
        - It's the job of the inspection team to do more than just identify the problems; they must also come up with the solutions.
    - The moderator compiles all the defect resolutions into an inspection log.

(Stellman & Greene, 2006) Page 74.

In this project, though, inspections will be used for the software requirements specification.

3.2. Walkthroughs

A *walkthrough* is an informal way to present technical documents in a meeting, the author creates a meeting with the reviewers, normally with presentation.
Walkthroughs are used when the author need the perspective from others without technical knowledges.

- Walkthroughs are used when the author of a work product needs to consider the perspective of someone who does not have the technical expertise to review the document.
- After the meeting, the author should follow up with individual attendees who may have had additional information or insights. The document should then be corrected to reflect any issues that were raised.

Walkthroughs will not be used in this project.

3.3. Deskchecks

A *deskcheck* is a simple review in which the author of a work distributes it to one of more reviewers, the reviewers read it and send the author the errors and comments. Deskchecks are made when a full inspection is not necessary.

- Unlike an inspection, a deskcheck does not produce written logs which can be archived with the document for later reference.
- Deskchecks can be used as predecessors to inspections.
    - Frequently, having an author of a work product pass his work to a peer for an informal review will significantly reduce the amount of effort involved in the inspection.

(Stellman & Greene, 2006) Page 84.

Deskchecks will be used in this project for the reviews (e.g., source code documents and reports etc.)

4. Tests

4.1. Unit Testing

Unit testing describes a testing method of checking small pieces or individual units of software. Therefore, unit tests will be achieved by testing the individual functions of our services (e.g., functions for creating a card, etc.).

As JavaScript will be mostly used for this project, the **Jest testing framework** will be used to build a testing pipeline for each function. The tests can be run manually on each contributor's devices and will run automatically on GitHub by using GitHub Actions to configure a Continuous Integration workflow to ensure the validity and correctness of the project. The workflow will be configured as such that every time someone pushes changes to a branch; GitHub will be triggered to run all the tests and show if there are any failing tests.

4.2. Integration Testing

Integration testing describes a method by testing independently developed units of software and if they work correctly when they are connected to each other.
The **Angular framework**, which will be used for this project, supports test-driven development out of the box, so integration tests can be written smoothly using the framework to ensure the units work well together. Also, those tests can be run individually on a device and will be automatically run by configuring the pipeline on GitHub.

4.3. Acceptance Testing

Acceptance testing eventually finishes the testing procedure by allowing a user to interact with the software with the aim of checking if expectations are matched with the delivered software. This decides if the software can be deployed in a productive environment.
Usually, acceptance tests are done by clients as they decide if software can be deployed. To ensure the correct behavior, E2E testing can be applied as well, which tests the workflow of the application from start to finish (testing actions that real users can perform on the application to check that desired outputs are obtained). This can also be done automatically by using continuous integration provided by GitHub.
For this project, the acceptance tests will be done manually by the team members to ensure the acceptance criteria are met.
Following you can find the links for the:

- Test plan
- Test report

5. Risk Management

5.1. Risk identification and assessment

For the top risks, with

Probability x Impact ≥ 15

potential indicators and actions have been described.
Link to the risk.

5.2. Risk monitoring and control

Risks are reviewed every week.
Risks with P x I ≥ 20 must be mitigated.

6. Coding Standards

Since the Angular Framework provides a page with official standards for the language, we are going to follow those. They can be found here. The team agreed to follow the following rules:

- Rule of one: every file must define just one thing, one component or one service.
- Limit files to ~ 400 lines of code to increase readability.
- Functions should not be longer than 75 lines, small functions are easier to test.
- Use naming patterns, like *feature.type.ts.*
- Use dashes to separate words in a name, dots to separate descriptions of the name.
- Upper camel case will be used for class names, camel case for objects.
- Put bootstrapping and platform logic for the application in a file named *main.ts*.
- Use *dashed-case* for naming the components.
- Do name test specification files the same as the component they test.
- Keep a flat folder structure if possible, creating sub-folders when a folder contains more than 7 files.
- Place properties up top followed by methods.
- Place private members after public members, alphabetized

7. Quality metrics

Flashee is a light application that must run smoothly so the user is comfortable while using it. Metrics will be divided into 4 types, UX metrics, performance metrics, project metrics and internal metrics for the code. UX metrics will evaluate the user interaction, performance metrics the use of computational resources and project metrics measure if the project is running according to the plan and to evaluate the quality of the project management.

1. **Usability metrics.**
   1.1. Loading times, how much it takes the program to load every screen, must be under a second.
   1.2. User errors, the number of times a user fails in archiving a task because a confusing design, resulting in an unpredicted outcome for the user. At the end of the project must be 3% to ensure usability.

1.3. How many times the application fails because of internal errors, ideally should be 0 but a more realistic number would be 1/1000 times a task is performed.

2. Performance metrics.

2.1. Use of CPU, functions will be analyzed to determine its O() complexity, functions must have a complexity lower then $O(n^2)$.

3. Project metrics.

3.1. Number of document corrections, a high number of corrections means there are issues with the project that can lead to high time spending, we want to make them lower or equal to 1.

3.2. Number of times that a component gets tested, ideally every component should be tested 3 times, more times can mean poor understanding of the code.

4. Internal quality metrics.

4.1. Portability, we will measure in how many browsers support the app, must work at least in 4 different browsers.

Quality goal	Metric
1. Usability	1.1,1.2, 1.3
2. Portability	4.1
3. Performance	2.1

Project metrics are inherent to the project, so they don't need a specific goal.