deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

TQS: Quality Assurance manual

Andreia Portela [97953]

Ricardo Ferreira [98411]

Tiago Matos [98134]

Vitor Dias [98396]

v2022-06-17

Index

1 Project management

1.1 Team and roles

Role	Assignee
Team Coordinator	Tiago Matos
Product Owner	Ricardo Ferreira
QA Engineer	Andreia Portela
DevOps Master	Vítor Dias
Developer	Everyone

1.2 Agile backlog management and work assignment

At the beginning of every iteration we had a meeting where the team coordinator would assign tasks and explain what was to be done during the week. This division was displayed in the management tool Jira.

We were working with user stories and user points and aimed to, at the end of each iteration, have a burndown chart that gets to zero (meaning that every planned task had been completed that week). If scheduled tasks weren't completed within the deadline, they would be added to the next iteration.

2 Code quality management

2.1 Guidelines for contributors (coding style)

Some guidelines to follow when coding:

- Comment the code (we are coding so that multiple people can read and understand the code)
- Keep methods short
- Use intuitive names for variables and methods
- Use the correct indentation
- Limit the scope of variables (for security measures)
- Avoid repetitions (it's better to create a new method than repeating lines and lines of code)
- Use Sonar Lint to avoid bugs

2.2 Code quality metrics

The main branch in GitHub is protected, meaning that in order to send the code to the main, a pull request must be created. This pull request can only be accepted with the following conditions:

- Attains the quality metrics assigned by Sonar (at least 80% coverage, less than 5% duplications, and grade A in security, reliability, and maintainability)
- Unity tests required (goes back to Sonar's coverage)
- Code is peer-reviewed (ensure it follows the code guidelines)
- The user story is complete (ex.: a functionality has been fully implemented)

This was implemented using GitHub Actions and creating a workflow in order to use Sonar.

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

To ensure a development focused on the user stories, we assigned to each git branch the user story name and tag attributed by Jira. We did the same with commits. This way, every task done in order to complete a user story would be saved in a single branch and it would be possible to view the steps followed to complete said user story. It also helps us see in Jira what is being done, and understand how the team is working.

After pushing the code to GitHub and opening a pull request, the code was reviewed by a peer, optimally one that was working with similar tools or tasks (to facilitate comprehension).

The team's “Definition of Done” includes the following:

- Code is up to the quality metrics
- Product owner accepts the user story
- User story acceptance criteria are met

Only when this DoD is met can the Pull Request be validated by the peer-reviewer and merged with the main.

3.2 CI/CD pipeline and tools

For the continuous integration pipeline, we used Github. In there we added a SonarCloud bot that would review the pull request and warn about bugs, vulnerabilities, etc in that commit. If the code wasn’t according to SonarCloud acceptance metrics, the Pull Request could not be accepted. Additionally, we decided to use SonarLint, a tool that detects bugs locally, to avoid multiple failed commits and keep the branches clean.

The continuous delivery was made in a Virtual Machine with 4 containers (2 for the databases, 2 for the servers). The code pushed to the GitHub main branch is directly sent to the VM, assuring the VM is always up to date with the latest version of the software.

4 Software testing

4.1 Overall strategy for testing

The team strategy for testing was test-driven (TDD). We started by writing unit tests and then the code itself. Behavioral tests were done using Cucumber (based on the user stories) and Selenium. Integration tests were also included to ensure good behavior between modules.

4.2 Functional testing/acceptance

As mentioned earlier, functional testing is to be done using Cucumber and Selenium. Each user story must be mapped into a Cucumber Feature and different scenarios, correctly identified. They are also summarized in the corresponding Jira issues. The steps in these scenarios must be written using Selenium.

4.3 Unit tests

Each developer designed the unit tests needed to complete their user story. This is, since the developer understood precisely what was asked of them, they would elaborate tests that would give the result required and then write the code to pass said tests.

With this strategy, the developer is sure that the code they write will have the expected results and validate the user story.

Using Sonar allows us to confirm that enough and complete tests have been written (through coverage), ensuring a thoroughly tested software.

4.4 System and integration testing

Since we did functional testing,i.e. we tested how the applications worked through the frontend, there were only a few things left to test in the integration tests. We tried to test things not yet tested with the functional tests like the behavior between modules (i.e. controller, service, and repositories/API).