universidade de aveiro
departamento de eletrónica, telecomunicações e informática

TQS: Quality Assurance manual

Afonso Campos [100055], Diana Siso [98607], Isabel Rosário [93343], Miguel Ferreira [98599]
v2020-05-25

[This report should be written for new members coming to the project and needing to learn what are the QA practices defined. Provide concise, but informative content, allowing other software engineers to understand the practices and quickly access the resources.
Tips on the expected content, along the document, are meant to be removed.
You may use English or Portuguese; do not mix.]

1 Project management

1.1 Team and roles

For this project, Isabel Rosário assumed the role of Team Coordinator, Diana Siso was the Product Owner, Miguel Ferreira was the QA Engineer and Afonso Campos filled out the role of DevOps Master. All of the team members worked as Developers.

1.2 Agile backlog management and work assignment

Concerning the agile backlog management practices of the work group, the team is using the Jira tool to keep track of the progress of the multiple tasks that must be completed for the construction of a minimal viable product.

The Team Coordinator created the project board on this platform and, as the project unfolds, creates and edits the tasks as well, ensuring they are each in the correct work column. It is also the Team Coordinator's job to assign the tasks to the various Developers and make sure their progress and completion is going according to the established timeline for the project. It is the responsibility of each Developer to move the tasks assigned to themselves to the In Progress or Done columns, according to their own work.

The Jira tasks are user story oriented, and it is advisable that each member completes a task before taking on the next one. If one Developer ever takes on too many tasks and fails to complete any of them, it is up to the Team Coordinator to notice and bring attention to the situation, as well as define a plan to bring the organizational system back on track.

The link to the group's Jira board is the following:
https://chateau-du-vin.atlassian.net/jira/software/projects/CHATEAU/boards/1

[Description of agile practices defined in the project for backlog management (user stories oriented) and job assignment, and links to associated resources. cfr. PivotalTracker workflow]

2 Code quality management

2.1 Guidelines for contributors (coding style)

- Don't catch generic exceptions: it obscures the reason for failure, try to make exception handling as specific as possible.
- Fully qualify imports: use `import foo.Bar;` instead of `import foo.*;`.
- Never use deprecated Java libraries.
- Use Javadoc standard comments.
- Define fields in standard places: define fields either at the top of the file or immediately before the methods that use them.
- Limit variable scope: keep the scope of local variables to a minimum.
- Use consistent indentation.
- Limit line length.
- Use *TODO* comments.
- Use logs – but sparingly.

[Definition of coding style adopted. → e.g.: AOS project]

2.2 Code quality metrics

Since we're using two public repositories, one for the management app and other for the store app, to analyze our code we'll be using Sonar Cloud. The projects will be analyzed individually, through an automatic github action that triggers the analysis for each push and pull request. The quality gate used will be the default Quality Gate set by Sonar Cloud because we think that it meets the project scope. As this analysis will be ran with every push to the repository, the developer will be able to better control the quality of his code, because the "New Code" analysis will be run by Sonar Cloud.

[Description of practices defined in the project for *static code analysis* and associated resources.]
[Which quality gates were defined? What was the rationale?]

3 Continuous delivery pipeline (CI/CD)

3.1 Development workflow

For the development of this project we decided to adopt a Git Feature Branch Workflow.
With that said, we'll have a ***main*** branch, containing a stable version of the product that will be updated only when there's a new version totally completed and a ***develop*** branch that contains the version of the application that is currently being developed.
Because we're following a *Feature Branch Workflow*, we'll create a branch everytime we want to develop a new feature for the product.
The branches will be named accordingly to the part of the project we're working (e.g.,*backend-creating_models*, *frontend-creating_home_page*, *deployment-creating_docker_file*, *ci-setting-sonar-cloud, etc…*).
Everytime a feature is done and tested, the developer makes a pull request from the feature branch to the *develop* branch that must be reviewed and accepted by the QA Engineer. When the pull requests are made by the QA Engineer we established that it should be the Team Leader reviewing and accepting the pull requests. When all the features planned for the version being developed are complete and tested, a pull request is made to the *main* branch.
Because the Jira issues are mapped to user stories, they are only complete when a user story is totally complete.
For them to be really completed, they must be fully implemented and tested.

[Clarify the workflow adopted [e.g.. gitflow workflow, github flow . How do they map to the user stories?]

[Description of the practices defined in the project for *code review* and associated resources.]

[What is your team "Definition of done" for a user story?]

3.2 CI/CD pipeline and tools

Since we're using GitHub as our code repository, we'll use Github Actions which is GitHub's own CI/CD platform. For the ManagementApp we will have a *build.yml* file that will trigger two actions:

- Running the ManagementApp Docker Compose and launch the Maven tests;
- Running a code analysis on Sonar Cloud and verify if the code passes the defined quality gate;

For the StoreApp, we'll also have a *build.yml* that will trigger the same actions as the ones described above but this time for the StoreApp Docker Compose file.
These actions will be run every time there's a *push* to the specific repository.
[We still don't have CD thought and defined]

[Description of the practices defined in the project for the continuous integration of increments and associated resources. Provide details on the tools setup and config.]

[Description of practices for continuous delivery, likely to be based on *containers*]

3.3 Artifacts repository [Optional]

[write here]

[Description of the practices defined in the project for local management of Maven *artifacts* and associated resources. E.g.: artifactory]

4 Software testing

4.1 Overall strategy for testing

[write here]

[what was the overall test development strategy? E.g.: did you do TDD? Did you choose to use Cucumber and BDD? Did you mix different testing tools, like REST-Assured and Cucumber?...]

[it is not to write here the contents of the tests, but to explain the policies/practices adopted and generate evidence that the test results are being considered in the IC process.]

4.2 Functional testing/acceptance

[write here]

[Project policy for writing functional tests (closed box, user perspective) and associated resources.]

4.3 Unit tests

[write here]

[Project policy for writing unit tests (open box, developer perspective) and associated resources.]

4.4 System and integration testing

[write here]

[Project policy for writing integration tests (open or closed box, developer perspective) and associated resources.]
API testing

4.5 Performance testing [Optional]

[write here]

[Project policy for writing performance tests and associated resources.]