TQS: Quality Assurance manual

Gonçalo Lopes [107572], Rodrigo Graça [107634], Tiago Cruz [108615]

[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

Team Coordinator: Tiago Cruz

Ensure that there is a fair distribution of tasks and that members work according to the plan. Actively promote the best collaboration in the team and take the initiative to address problems that may arise. Ensure that the requested project outcomes are delivered on time.

QA Engineer: Gonçalo Lopes

Responsible, in articulation with other roles, to promote the quality assurance practices and put in practice instruments to measure que quality of the deployment.

Monitors that team follows agreed QA practices.

DevOps master: Rodrigo Graça

Responsible for the (development and production) infrastructure and required configurations. Ensures that the development framework works properly. Leads the preparing the deployment machine(s)/containers, git repository, cloud infrastructure, databases operations, etc.

Developer: All elements
Responsible for monitoring the pull requests/commits in the team repository.

1.2 Agile backlog management and work assignment

In the **newPhi** project, our team employs a comprehensive Agile backlog management and work assignment process designed to optimize efficiency, adaptability, and team collaboration. The approach is fundamentally user story-oriented, ensuring that all functionality delivers tangible value to the users and aligns closely with their needs.

Backlog Management Practices

- **User Stories Creation**: All features and requirements are broken down into user stories that describe functionality from the perspective of an end-user. This helps keep the focus on delivering value directly observable by the users.
- **User Stories Grooming**: Regular backlog grooming sessions are held to refine user stories, ensuring they are clearly understood and actionable. This includes defining acceptance criteria, estimating effort, and identifying any dependencies or blockers.
- **Prioritization**: User stories are prioritized based on various factors such as business value, user impact, and technical complexity.
- **Sprint Planning**: At the beginning of each sprint, the team plans which user stories are selected from the top of the prioritized backlog.
- **Tracking Progress**: During the sprint, progress on user stories is tracked, allowing all team members to see the status of tasks in real time.

Work Assignment Practices

- **Team Roles**: Each team member has a defined role (e.g., developer, tester, UX designer) but is encouraged to take ownership of tasks across the spectrum to promote skill development and improve project understanding.
- **Task Ownership**: Tasks are not assigned by the project manager but chosen by team members themselves based on interest and expertise. This self-assignment method increases engagement and accountability.
- **Daily Stand-ups**: Daily stand-up meetings are conducted where team members discuss what they did the previous day, what they plan to do today, and any impediments they are facing. This facilitates quick resolution of blockers and keeps the team aligned.
- **Pair Programming**: For complex user stories or to foster knowledge sharing, pair programming is encouraged. This practice not only improves code quality but also helps in disseminating knowledge across the team.
- **Retrospectives**: At the end of each sprint, a retrospective meeting is held to discuss what went well, what could be improved, and how the team can adjust its practices to increase productivity and work satisfaction.

Tools Used

- **JIRA**: For backlog management, sprint planning, and progress tracking. It allows for detailed reports and real-time collaboration.

2 Code quality management

2.1 Guidelines for contributors (coding style)

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

2.2 Code quality metrics and dashboards

[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

[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

[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 System observability

3.4 Artifacts repository [Optional]

[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

[what was the overall test development strategy? E.g.: did you do TDD? Did you choose to use Cucumber and BDD? Did you mix sdifferent 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

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

4.3 Unit tests

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

4.4 System and integration testing

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

4.5 Performance testing [Optional]

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