# TQS24: Project Assignment Guidelines

*I. Oliveira, v2024-04-16*



# 1 Project framework

## 1.1 Project assignment objectives and assessment

Students should work in teams to specify and implement a medium-sized project, comprising:

- Development of a viable MVP (functional specification, system architecture and implementation), applying software enterprise architecture patterns.
- Specification and enforcement of a *Software Quality Assurance* (SQA) strategy, applied throughout the software engineering process.
- Apply the engineering practices of Continuous Testing, Continuous Integration and Continuous Delivery.

## 1.2 Business scenario

Consider the following hypothetical business scenario to illustrate the expected new digital services:
«The New Private Health Initiative (newPhi) needs an integrated IT solution to handle patient appointments and patient admissions at their future hospitals. For the moment, clinical records are out of scope of the project and the main focus is on pre-appointment (patients search for a consultation or exam), receptions services (register patient, waiting rooms, call patients) and post-encounter (settle bills, issue presence confirmation...).
According to exploratory requirements (further analysis is still needed), newPhi needs a platform that includes:

- Φ-Patient: customer portal in which clients can schedule new appointments, check their "agenda" and browse care file (previous appointments). Robust scheduling is an essential requirement to offer competitive services (e.g.: search by medical specialty, prefer a hospital but consider nearby options, ...).
- Φ-Desk: registration desk and staff members services to check-in patients, handle registration and administrative tasks (payments, ...). Staff can check waiting lines and call for awaiting patients, either for the Reception desk, or for the Consultation / Examination offices.
- Φ-Boards: simple digital signage solution to handle the call screens at the Hospital facilities (Figure 1). Usually, lines are processed by order of arrival, but there are priority lines too.

Later in the project, the newPhi expects to offer a mobile app, similar to the Φ-Patient portal. Patients should also be available to self-check-in, if they wish, when arriving at the hospital.»

Note that this is an overview of the business context; your group can adapt and introduce different flavors/innovations and **make the generic business case more tangible/realistic**.
**Variations on the theme are acceptable** if you keep the overall system architecture (see 1.3), e.g.:

- A solution to buy bus tickets (book seats), including the support for the bus operations staff (e.g.: defining trips, configure availability), and call screens at stations (e.g.: informing bus arrivals and on-boarding).

Note that a full-featured solution for the above scenarios is out of scope of the course! You should **prioritize services to deliver a meaningful MVP** at the end of the semester.
You will likely invest in the backend but, for example, choose a simple approach for the digital signage component. Check with your teachers.

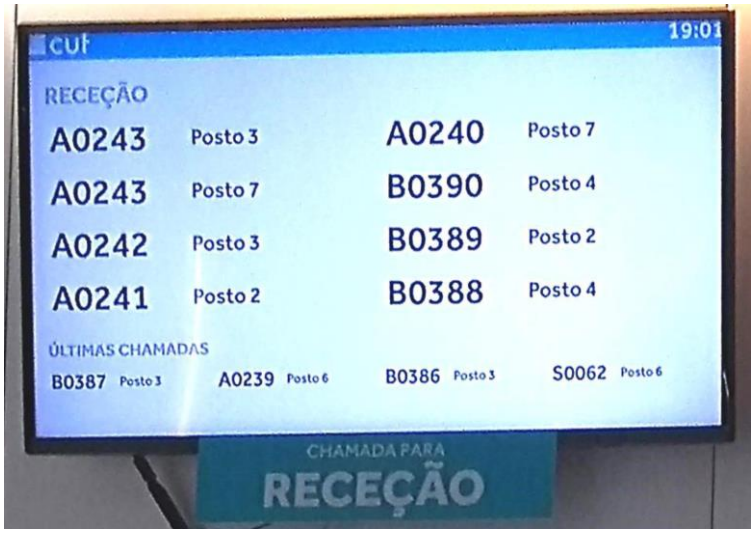

Figure 1: A call screen at a hospital facility.

## 1.3 Reference architecture and constraints

The groups should specify their own system architecture, but consider the following system constraints:

- Customer Portal and Registration Desk are different web apps that will likely share the same backend.
- Backend components are to be implemented in Spring Boot.

- The Digital Signage (DS) can be very simple for now but should be designed as independently as possible, so it can be explored independently in the future.
- The **Mobile app is optional**; however, the architecture should be designed to fully support/expect a future mobile app.
- All implemented components should have a QA/testing. For frontends, UAT is expected; for the core backend, a multi-layer and comprehensive testing strategy should be implemented.

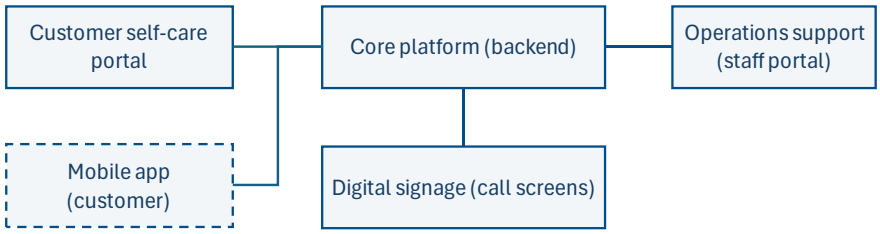


**Figure 2: High-level architectural blocks. (Implementation details omitted on purpose.)**

# 2 Project management and schedule

## 2.1 Agile project management

Teams should adopt an agile development approach and corresponding **agile project management** practices. Agile project management uses iterative development (key value is identified as epics, the project is divided in sprints that deliver a focused increment), plans and visualizes work as user stories (small but recognizable product features) and adapts/ prioritizes according to business value.

**Stories have points**, which reveal the shared expectation about the effort the team plans for the story, and prioritized, at least, for the current iteration. Developers start work on the stories on the top of the current iteration queue, adopting an agreed workflow.

User stories act as the unit of work acceptance and provide the natural context for validation given **the acceptance criteria**, as used in a Behavior-driven approach.

- ➔ Practice: Agile Project Management.
- ➔ Suggested tools: JIRA (with Scrum project template).
- ➔ Practice: business-facing acceptance criteria as scenarios with examples

## 2.2 Team and roles

All students need to contribute as *developers* to the solution. Each team/group should assign additional roles:

| Role | Responsibilities |
|---|---|
| Team Coordinator (a.k.a. *Team Leader*) | 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 in time. |
| Product owner | Represents the interests of the stakeholders.<br>Has a deep understand of the product and the application domain; the team will turn to the Product Owner to clarify the questions about expected product features.<br>Should be involved in accepting the solution increments. |
| QA Engineer | Responsible, in articulation with other roles, to promote the quality assurance practices and put in practice instruments to measure que quality of the deployment.<br>Monitors that team follows agreed QA practices. |

| Role | Responsibilities |
|---|---|
| DevOps master | 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 members contribute to the development tasks which can be tracked by monitoring the pull requests/commits in the team repository. |

## 2.3 Reference iterations plan

The project will be developed in **1-week iterations**. Active management of the product backlog will act as the main source of progress tracking and work assignment. Each "Prática" class will be used to monitor the progress of each iteration; column on the right lists the minimal outcomes that you should prepare to show in class.

Expected results from project iterations:

| **Iter. # (start)** | **Main focus/activities for the iteration** | **Results to be presented in this week class (*Práticas*)** |
|---|---|---|
| **I1**, prep 18/04 | • Define the product concept. Work on Personas, main scenarios, and expected Epics.<br>• Team resources setup: code repository, shared documents, team development workflow,...<br>• Start backlog. | n/a |
| **I2** 25/04 ([a]) | • Define system architecture.<br>• Define the SQE tools and practices (initial version).<br>• Product specification report (draft version; product concept & requirements section)<br>• Backlog management system setup.<br>• UI prototyping (partial) | Outcomes from I1:<br>• Product concept.<br>• Software/system architecture proposal. |
| **02/05** | [Academic week] | |
| **I3** 09/05 | • A couple of core user stories detailed and implemented.<br>• Concept for the call board defined.<br>• QA Manual (report)<br>• CI Pipeline (initial version). | • Present your SQE strategy.<br>• UI prototypes[1] for the core user stories.<br>• Plan for next iteration: user stories to implement. |
| **I4** 16/05 | • Develop a few core user stories involving data access & persistence (for customer and staff).<br>• Comprehensive API.<br>• Set up the CD pipeline. | • Product increment with (at least) a couple of core user stories.<br>• "Live" call screen.<br>• CI Pipeline, QG and merge-request policy.<br>• Plan for next iteration: user stories to address and implement. |

[1] These "prototypes" would be early versions of the web pages, already implemented with the target technologies (and not mockups) but more or less "static" (not yet integrated with the backend/business logic).

| Iter. # (start) | Main focus/activities for the iteration | Results to be presented in this week class (*Práticas*) |
|---|---|---|
| **I5**<br>23/5 | • Stabilize the Minimal Viable Product (MVP).<br>• All deployments are available in the server.<br>• Relevant/representative data included in the repositories (not a "clean state").<br>• Product specification report (final version.)<br>• Non-functional tests and systems observability. | • Comprehensive REST API.<br>• CD Pipeline: services deployed to containers (or cloud).<br>• Quality dashboard. |
| 30/05 (b) | • Minimal Viable Product (MVP) backend deployed in the server (or cloud). | Oral presentation/defense. |

(a) 25/4 is a holiday; still, consider the weekly duration for iterations, i.e., the teams should start Iter. 2.
(b) The final presentation dates need to be confirmed with students.

# 3 Implementation guidelines

## 3.1 Team and coding practices

The team should select and apply a development Git workflow, such as "GitHub Flow" or "Gitflow workflow".
You are expected to complement this practice by issuing a "pull request" (a.k.a. merge request) to review, discuss and optionally integrate increments in the *master*. Static code analysis should be performed before reviewing a pull-request.

➔ Practice: Git workflows
➔ Practice: Integrate code reviews in the "pull request" analysis

## 3.2 Quality assurance and DevOps

### 3.2.1 Story driven

The user story is the unit for project tracking and acceptance.
A user story development approach plays well with a feature-branching Git workflow.

➔ Practice: story-driven development.

### 3.2.2 Style Guide

Each group should set its guidelines for contributors. This is not to be an extensive document, but rather a selection of key practices (pointing to other complementary references).

### 3.2.3 Test-driven

Software developers are expected to deliver both production code and testing code. The "definition of done", establishing the required conditions to accept an increment from a contributor, should define what kind of tests are required.
Some stories may be tailored for data-driven tests, which can be handled at different levels, e.g., Gerking Scenario Outlines; JUnit 5 capabilities for parameterized tests can be used where appropriate to increase the number of tests and therefore their coverage.
If you have a scenario with multiple input variables and their respective values, and you foresee some risk related with these variables and their interaction, consider testing combinations of these variables; as generating all

combinations can be unfeasible, consider all pairs of variables (pairwise-testing); you can use a tool such as PICT to generate a limited set of test scenarios that you can injest to your parameterized test as test data.

Although suggested, TDD is not mandatory. Teams should emphasize when and how they use TDD (in reporting).

➔ Practice: test automation for unit tests, integration tests and acceptance (with BDD)

#### 3.2.4 CI & CD and continuous testing

The team should define, setup and adopt a CI/CD pipeline. The project should offer evidence that the contributions by each member go through a CI pipeline, explicit including tests and quality gates assessment (static code analysis). The team should also offer CD, either on virtual machines or in cloud infrastructures.
To achieve effective continuous delivery (i.e., features delivery can be triggered at any time), the team should adopt the "infrastructure as code" principle, for test, stage and production environments,

➔ Practice: docker containers

➔ Practice: continuous integration and continuous delivery workflows

#### 3.2.5 QA Dashboards

QA oriented dashboards are required, including:

- code level decision dashboard (e.g.: SonarQube)
- requirements/stories level dashboard (e.g.: Xray plug-in for JIRA).

#### 3.2.6 System ops observability

Provide dashboards and alarms on the operation conditions of infrastructure and deployed services. Instrumentation to monitor the production environment (e.g.: Nagios alarms, *ELK stack* for application logs analysis, HTTP errors, performance degradation...)

### 3.3 Project outcomes/artifacts

#### 3.3.1 Project repository

A cloud-based **Git repository** for the project code and reporting. The main **submission method will be the git repository**.
Besides the code itself, teams are expected to include other project outcomes, such as requested documentation. The project must be shared with the faculty staff. Expected structure for main repo:

```
README.md
docs/
projX/
projY/
```

...with the following content:

- docs/ → reports should be included in this folder, as PDF files, especially the QA Manual and the Project Specification report. Presentation support should also be copied here.
- projX/ → the source code for subproject "projX", etc.
- **README.md** → be sure to include an informative README with the sections:
  a) Project abstract: title and concise description of the project.
  b) Project team: students' identification (and the assigned roles, if applicable).
  c) Project bookmarks: link **all relevant resources** here!
    - Project Backlog (e.g.: JIRA, GitLab agile planning)

- Related repositories (there may be other related code repositories…)
- Related workspaces in a cloud drives, if applicable (e.g.: shared Google Drive)
- API documentation (link the landing page for your API documentation)
- Static analysis (quick access to the quality dashboard)
- CI/CD environment (quick access to the results dashboard)
- …

For certain CI/CD pipelines, it could be **better to use more than one repo**, i.e., specific git repositories for different projects/modules. In that case, consider:

- using a "main repository" and be sure to link related repositories in the README.md, or
- create a "group" of repositories (if your cloud infrastructure has that concept) and share the group.

### 3.3.2 Project reporting and technical specifications

The project documentation should be kept in the master branch of the repository (folder: /docs) and updated accordingly.
There are two main reports to be (incrementally) prepared:

1. Product Specification report [→ template available in the project resources]
2. QA Manual [→ template available in the project resources]

### 3.3.3 API documentation

Provide an autonomous report (or a web page) describing the services API. This should explain the overall organization, available methods and the expected usage. See related example.
Consider using the OpenAPI/Swagger framework, Postman documentation, or similar, to create the API documentation.