Quality Assurance Plan

Sustainable Work through Women-in-tech Application for Older Women in Malaysia and Thailand: Integrating Action Research and Design Science Approach

1. Introduction

The purpose of this document is to discuss the process flow that the team will follow to avoid defects from appearing during the development of the software application.

Each team will have their own specified repository that will be used within the respective team for them to work on with their shared tasks. After the implementation of each feature, the Release Train Engineers (RTE) of the team will review and merge the team branch to the master repository.

During the implementation of the project, the members will develop tests whenever necessary to ensure that the code has no errors. These tests will be automatically run with each push to the repository by the CI/CD pipeline that is offered in GitLab. If these tests fail, the members will be notified that there is an error present in the code.

Some features will have its set non-functional requirements that can't be easily tested with a unit test code. These features will be included in the definition of done that the RTE will use to review the code.

At the end of each iteration, the code artifacts will be the source code that forms the application and the unit test files that will be used to test the source code.

2. Git Repository and Branching Model

Since every team will be sharing the repository, a standardized process will have to be followed by every member working in the project to ensure the chances of conflicts happening are minimized. In general, each team will have a branch of their own to allow for good development efficiency since every team handles different parts of the software. Each branch will be responsible for only one feature. This means that if a team is developing multiple features in one iteration, they will have multiple branches that they will be responsible for. The work done by each team will only be merged into the master branch after a review has been conducted on code developed and verified the work done satisfies the requirements.

3. Commits

3.1 Commit frequency

Each developer will be required to commit to their branch frequently so that other developers have a good grasp on the overall progress of the feature or the user story that is being worked on. However, the developers should limit the commit to when a part of the feature is completed rather than when a line of code has been written.

Every commit to the main branch should be independent of other commits. Meaning that each branch is only allowed to be responsible for one feature.

3.2 Message style

Commit messages should contain short and concise messages that are contained within 50 characters. Each message should contain the purpose or details of the commit. For example; Login feature - UI interface.

3.3 Tags

Annotated tags should be used to mark the stable version of a working feature when required and Lightweight tags should be used to mark commits that introduce some unexpected behaviour or bug. The lightweight tags must be removed when the bugs are resolved.

4. Branch

4.1 Naming conventions

Each branch should contain the name of the team that is responsible for it and the feature that is being developed under that branch. For example; Team RedCow - Login.

4.2 Branch creation and deletion

Branches should only be created when there is a new feature that is being developed or when there is a bug/unexpected behaviour from certain features that would require it to be resolved in another working environment. We will also create branches when we need to have a spike to test out new approaches in the development of the application.

When a branch is merged with the main branch and has served its original purpose, the branch will then be deleted.

4.3 Branch protection

Every branch will be locked and protected to the developers that are related to it to ensure that unrelated developers don't accidentally make changes to the branch. Access to the branches will be given by the RTE of the team.

5. Merge Requests

5.1 Merge request handler

With smaller features, RTE and the product owner of each team should be in charge of reviewing the merge requests as they have the best understanding of the client's needs and thus should be able to identify whether the branch meets the requirements.

With bigger features, the team should hold a meeting to discuss the merge request's appropriation.

5.2 Testing with CI/CD

Every team should develop a set of tests that will be automatically executed whenever a commit is made to the branch so that basic errors are avoided. This will also help in reducing the load of the merge request handlers.

When merging to the main branch, the merge request handlers should manually review the code to ensure a proper integration to the main system and there are no new errors that are introduced in the process of merging.

5.3 Giving labels

Labels should be added to merge requests where appropriate to allow for a smoother workflow for the reviewer and the team. These labels can be defined previously or newly created to categorize the type of merge requests. The team should ensure that the labels that are created are given a detailed description that contain the purpose of it and where it can be applied.

6. Rules to Follow

Code	Subject	Description	If violated
C1.1	Commit	Only merge to the main branch when a user story or feature has been completed or updated	RTE contacts that team member to reverse the merge
C1.2	Commit	Add lightweight tags on commits that introduce bugs or unexpected behaviour	RTE adds the lightweight tags
C1.3	Commit	Make stable version of the software using annotated tags	RTE adds the annotated tag
C1.4	Commit	Ensure that each commit has message with format discussed	Committer changes the commit message using the --amend command and force pushes it as soon as possible
B1.1	Branch	Name branches using the naming format discussed	Branch creator renames the branch
B1.2	Branch	Only give permission(push/merge) of a branch to related developers	Maintainer removes permission(push/merge) of unrelated developers
B1.3	Branch	Each branch should only be responsible for one feature	Team creates a new branch and transfer the codes for the new feature to that branch
B1.4	Branch	Only create branch under the circumstances discussed	RTE removes the branch created
M1.1	Merge Request	Merge requests must be tested with predetermined jobs and reviewed	RTE reverts the merge request