**Testing**

The aim of testing is to program the software as we planned and potentially prevent it from being broken. The following parts will introduce how the team deployed unit testing, integration testing, release testing, acceptance testing and continuous integration during the development.

**Unit Testing**

As TDD instructs, developers in the team wrote unit tests for most basic software components before coding any actual functionalities. Therefore, unit tests work as the base of the whole project. By doing unit testing, the team has a clearer view of what features a component is expected to achieve.

All of the unit tests were firstly planned by documenting test plans in detail. Any failed case and modification were also recorded in a test log for future bug track convenience. Detailed test plans and logs can be viewed in appendix X.

文本

描述已自动生成

Figure1. Unit testing & Integration testing

**Integration testing**

Integration testing tests subsystems [1]. In this project, scenes and huge combinations of multiple components are considered subsystems. This is relatively helpful to check whether a subsystem works as the specification expected. Integration testing is conducted along unit testing. Results and logs are in appendix X.

Examples of Integration testing are as follows:

1. Snapshot created and match with old one.

2. Test interactions manually in a subsystem to check them meet the requirement.

**Release testing**

Release testing is expected to be conducted by an individual quality assurance team that has not been involved in the system development \cite{reqVSspe}. However, due to the team's small size, all the team members have done something related to the system. In this case, two members who focus more on user interface would take responsibility for release testing. They tested the software as a whole system manually to check whether the system achieves all the specifications and works without abnormal.

Three strategies taken are as follows:

1. Performance driven.

2. Specification driven.

3. Scenario driven.

**Acceptance testing**

Acceptance testing is the test after release testing and is done with the clients. It aims to check whether the software meets stakeholders’ expectations and receive feedbacks from them \cite{reqVSspe}. Team 10 prepared checklist using similar strategy as release testing’s and invited 6 stakeholders from year 2 and year 4 and Dr Heshan to do the acceptance testing. Team 10 showed the software to them and asked them to try the software freely. After trying the software, Team 10 discussed the problems with stakeholders and took notes of their comments and feedbacks, which can be seen in appendix.

**Continuous Integration**

Continuous integration (CI) suggests that all code changes will be processed in the mainline of version control to build and test the software automatically \cite{reqVSspe}. This approach supports TDD well since each submission will be built and tested on the server, which enforces testing and ensures all the tests pass before coding new features. Since all the tests will be run, it could prevent previous work from being broken from new changes, and bugs could be identified quickly.

图形用户界面, 文本, 应用程序

描述已自动生成

Figure2. Continuous integration

References (this would be formalised later)

1. <https://reactjs.org/>
2. <https://jestjs.io/>
3. <https://testing-library.com/docs/react-testing-library/intro/>

@misc{React,

howpublished = "\url{https://reactjs.org/}",

title = {React - A JavaScript library for building user interfaces},

year = {2021},

author = {Facebook Inc.},

note = "Accessed: 2021-3-25"

}

@misc{Jest,

howpublished = "\url{https://jestjs.io/}",

title = {Jest - delightful JavaScript Testing},

year = {2021},

author = {Facebook, Inc.},

note = "Accessed: 2021-3-25"

}

@misc{ReactTestingLibrary,

howpublished = "\url{ https://testing-library.com/docs/react-testing-library/intro/}",

title = {React Testing Library},

year = {2021},

author = { Kent C. Dodds and contributors },

note = "Accessed: 2021-3-25"

}