Evaluation & Testing

Within the product development process, it is important for to evaluate how well the final product fits the brief to ensure the product is useful for its intended purpose. It is also important to test the final product, which means that the code must be of an appropriate quality.

Evaluation

Product evaluation involves determining if the final product meets its brief, to do this we used the requirements that had been produced and adapted throughout the assessments.

We produced a set of specific, detailed requirements for the initial brief, which were used during product development to ensure the game did what it was supposed to. These requirements are known to meet the brief, making them useful for evaluating the final product.

Due to the changes introduced to the brief in Assessment 4, the requirements produced for earlier assessments were modified and adapted to fulfil the brief. The changes include requirements [2.1.5, 4.1.1, 4.1.4, 4.1.5]. These can be found in the requirements document on our website. The code was adapted accordingly.

Using the requirements document, we evaluated our final product. To achieve this we met as a team, where we went through each requirement, and ensured it was properly implemented in the final product. This also involved using our acceptance tests to ensure that an end user can perform appropriate actions that prove the requirements have been met. See **Appendix D** for the acceptance tests.

As our code was written using the principles of test driven development, we wrote acceptance and unit tests before we started coding. We wrote the acceptance tests beforehand using the requirements as a guide - this proved helpful for our evaluation, as it meant we had a set of tests to run that ensure our final product meets its brief.

The acceptance tests allowed us to evaluate our final product effectively as they allowed us to ensure the solution implemented in our final product performed as expected, and that the associated requirements were met.

Although acceptance tests were useful for evaluating the final product, not all the requirements had associated acceptance tests, which meant we had to determine as a team whether the untested requirements were met. For example, we had to manually determine that this requirement was met: [1.1.4] Must run on university computers.

Testing

Our team focused on testing, which involved ensuring the quality of our final product was high throughout the assessments. We focused on setting high standards for the code we were writing, as well as ensuring the game was fully tested and maintainable – all qualities important for a software engineering product.

The approach we took to testing was consistent across every assessment completed by our team – the only addition in Assessments 3 & 4 was code coverage. Other than that, we stuck to the same tooling and methodology as we felt it had been successful.

We wrote the game code using the principles of test driven development, using a combination of unit tests and end-to-end acceptance tests. The purpose of this testing was to ensure our code was of high quality – and using two different types of testing mechanisms helped with this. We had regular code reviews and continuous integration to aid with code quality, this is detailed later.

As a team, we determined appropriate quality code for our product to meet the following points:

- Code should function as expected
- Code should be covered by acceptance tests, to ensure it meets the requirements
- Code should be unit tested where appropriate, to ensure regressions do not occur
- Code should pass all automated unit tests
- Code should be reviewed by a developer not involved with writing it
- Code should be fully documented, including JavaDoc comments

Unit tests

Firstly, we wrote unit tests where possible before we started implementing features of the game. We found this highly useful for ensuring the code quality was high, because it forced us to think about the system design, as well as the function of the code – this includes thinking about inputs, outputs and API design before writing a single line of code. This approach involved ensuring our code functions as expected, and they tended to be written using a white-box approach. See **Appendix E** for the unit test listings.

In general, we enforced a development rule that meant all new code had to have unit tests, if achievable. Writing unit tests wasn't practical for every part of the codebase – especially UI orientated parts of the game. This is because often there are many ways of implementing the same thing, and we were unable to find an approach to testing the user interface with a reasonable amount of effort. We were limited in time, so there was a trade-off between amount of unit testing, and amount of progress on the product we had to consider. However, we used test-driven development with unit tests wherever it was reasonable to do so, and where it wouldn't take up too much time we could better spend on other aspects of the project.

Alongside writing unit tests, we set up a continuous integration server to run our tests every time a commit was made to our code repository. This was an added check to ensure we did not break anything in the codebase while implementing new features – this has helped ensure code quality is high, because it has allowed us to detect regressions and possible bugs in the code, with minimal effort after the initial implementation of the unit tests. Before committing any code to our "master" branch, we ensured all unit tests had passed on our continuous integration server.

Our continuous integration server also ran a script that determined code coverage – this is a key indicator that allows us to identify how much of the codebase is unit tested. We aimed to maximise our test coverage throughout the development process – we inherited a codebase with 20% coverage, and ended up with 33% coverage in our final product. The test report is available on our website. Ideally, we would have liked to increase that test coverage much more, however we were time limited and felt resources would be better spent elsewhere.

Code reviews

Our development process involved code reviews at every stage. This was a way of allowing our team to test and ensure the quality of code being written is high.

Each new feature or bug fix that was implemented in our product was coded normally by one or two individuals in a separate branch, a copy of the “master” branch of the code. Our “master” branch of code was the latest, working version of the game, and code could not be committed from branches into the “master” branch without having gone through a code review.

Our code reviews consisted of other team members commenting and discussing the contents of code awaiting a merge into master. We did this to ensure that new code is readable, maintainable and sensible – this helps ensure code quality is high because often when somebody who isn’t involved in the implementation of a feature sees the code, they spot mistakes or issues not noticed by the developer. This is beneficial, as it has caught issues and problems within the codebase that otherwise would have gone unnoticed, and has helped with the code structure as often reviewers would suggest better ways of doing things.

We would also block merging into the master branch if the unit tests had failed on continuous integration – this would normally be identified by the reviewer at the code review stage.

Acceptance tests

As mentioned earlier, we also wrote end-to-end acceptance tests to ensure that our final product met the requirements we had determined from the brief. These tests allowed us to ensure the user interface worked as expected, and to ensure that all requirements were being met by the implemented code.

Pair programming

Parts of the implementation was done using pair programming – where we had two individuals working together during the coding. The use of this method ensured that these key parts of code were reliable and efficient since there were two people solving problems rather than one.

We decided in group meetings whether to pair program a section of code or not. Pair programming also allowed team members to teach and learn from each which in turn led to better code with less bugs.

Code documentation

Throughout the development process, we focused on adding code comments and documentation of code base to ensure that other developers can pick up our code and be able to maintain it. We used JavaDocs for comment formatting - this proved useful because it allowed us to generate a HTML website containing structured and organised formatting of our code documentation, which we found helpful to reference while developing.

Overall we felt like the code quality and testing approaches we took in this project were successful towards ensuring a high standard of work, but as ever there are more things we could have done. Ideally, we would have liked to have achieved a greater test coverage overall, however we were time limited, and it wasn’t feasible to write extensive testing for the project we inherited in the time we were provided with.

Meeting the requirements

The product produced by our team meets almost all of the requirements determined from the brief given to us by the client. However, there are a number of minor requirements that weren't completed in time for the hand in, these are discussed later.

In general, we feel the product produced by our team has fulfilled the requirements set for the product - it is a usable, playable and enjoyable murder mystery game set in the Ron Cooke Hub. As requested, key functionality like dynamic storytelling have been implemented, along with a comprehensive selection of features including scoring, clues, characters, and additional items like the journal/inventory that aid the user experience.

The game we have built includes turn-based multiplayer support as requested - this was implemented in a flexible manner, initially to support two players. Due to the code structure of the implementation, this flexibility has given us the ability to have as many players as necessary with no negative effects on the gameplay, all by changing a single integer variable. We felt this was a good thing to show off, as it demonstrated that the codebase is quality and not prone to breaking with additional modifications and enhancements.

However, not all of the requirements our team set for ourselves were met. These are as follows:

- **1.3.1 Could be controlled by a gamepad**
    - This wasn't implemented in our final product due to lack of time, the team felt this feature didn't provide enough of a benefit with respect to the amount of work involved in implementing it. This requirement was always a nice-to-have thing and wasn't initially specified by the client, so we decided to leave it out of the final product for this reason.
- **6.2.1 There could be an online scoreboard to keep high scores**
    - As with the previous requirement, an online scoreboard would have been a bonus feature, and wasn't specified in the brief from our client. This requirement would have required a web service to be developed in order to store the scores online - our team felt this wasn't achievable in the time we had to develop the product.
- **7.1.7 The player must be shown introductory and closing dialogue**
    - This was a requirement we had implemented in our Assessment 3 project, which provided users with an introduction to the game and wrapped it up at the end. We intended to implement this requirement for Assessment 4, however we had to prioritise features, and came to the conclusion our time could be better spent elsewhere.

The full requirements document containing all of the requirements for the final product, including the un-implemented ones, can be found here.

Overall, our team is satisfied that the final product we have produced meets all of the core requirements of our client, despite missing a couple of nice-to-have features. Given more time, our team would have liked to add additional features and functionality to the game, to give it more polish and an improved user experience, however we are completely happy with the outcome of our final product, and are satisfied it meets the client's needs and requirements.