Test Evaluation

This document contains the testing evaluation for the ILP Pizza Dronz project. This will consider any inadequacy of the tests that were used, and identify what could have been improved given more time and better tools.

Gaps and omissions

There were several deficiencies in the testing process. Firstly, synthetic data was used throughout the testing process to simulate real data as closely as possible. However, as it is not real world data, it will not be able to simulate all possible real scenarios that could occur, and there may be instances that were not considered or overlooked. There may be some undiscovered cases that would cause the system to malfunction when used in a real setting. If given more time, the generation of synthetic data could be automated instead of manual, using a mocking framework such as EasyMock.

It was also difficult to test the non-functional requirements such as the performance of the system. For instance, whilst the time to calculate flight paths for all orders in a day did not exceed 60 seconds for any of the synthetic data that we had, we do not know how popular the service will actually be. Since this was only tested using the estimated average number of orders a day, there is not a high level of confidence that the system will always be able to generate the flightpaths this quickly.

There was difficulty involved with testing the maintainability of the code as well. While automatic formatters were used to neaten the code, and commenting was used throughout it is difficult to test how easily the system can be adapted in future, apart from perhaps by pier review from another developer.

Another gap found in testing was that the input data, either from manually synthesised mock data or from the REST server, was not always fully tested to ensure it was of the correct format. For example, the data for ‘restaurants’ gathered from the REST server would be assumed to be in the correct format. This could cause large problems if proper testing for this is not added in future. Additionally, the URL of the rest server was not exhaustively verified, and there may be some URL formats that would be allowed by the system but would cause many bugs in the program.

Target coverage/performance levels

There were many metrics used to evaluate coverage at different stages in development and different areas of the system. Firstly, the target method/function coverage should be at 90%, as this ensures that most methods were ran at least once during testing. This does not ensure that the system will be bug free, as there can still be inputs and scenarios that have not been covered. This is achievable given the smaller scale of the system.

For all written tests, the pass rate should be 100%. If a unit test or another written test fails, it most likely indicates that there is an error in the code, and that this will need to be rectified. If the pass rate is less than 100%, then the code will be examined and corrected appropriately.

The target branch coverage will vary between high and low level requirements, but should aim to be around 80%, as this ensures that a large amount of the paths in the code have been taken and tested.

The requirement coverage should be close to 100 percent but may not reach this. This is due to the non-functional requirements being difficult to test and sometimes having varying degrees of success. The functional requirements coverage should be 100%, as this is feasible with a project of this size and in the timeframe that we have to do it.

Coverage comparisons

Method coverage was calculated to be 94%. This is above the target coverage, and so gives strong evidence that most of the methods written have been used through the test suites. The method coverage only doesn't reach 100% as some of the methods were assumed correct through testing of other classes.

The pass rate was 100%, eventually. When the initial pass rates weren't 100%, the code was re-evaluated, and the bugs that the tests alluded to were fixed. After revisions to ensure the system was working the intended way, the pass rate reached its 100% mark.

Branch coverage varied widely from requirement to requirement, but never dropped below 72%, and this was for low level requirements so is considered enough to be confident with the system. The branch coverage of the whole system was 84%, which is above the target 80% and so we can be confident that a high level of the code decision branches were taken in our test suites.

Finally, the requirement coverage was 100% for the functional requirements. This was the target, and shows that we have achieved every functional requirement of the system specification. The non-functional requirements are more difficult to quantify, but all of them were achieved to some level of degree.

Improvements to achieving target levels

From our results, we have shown that the testing approach did reach the adequacy criteria that was set out. This does not mean that the system is bug free however, as there may be unexpected and untested input variations that will only be found when real world data is used. The code coverage is a good measurement and it helps to find any test suites that should be implemented, but it does not prove how effective the test suites we have are.

With more time, more test suites for edge cases should be generated to fully ensure that any bug in the system are caught, no matter how rarely they would cause an issue. This could be achieved by having the test suites be generated automatically, which would also eliminate human error in creating them, though this would take much more time and resources to set up.

Non-functional requirements could undergo more thorough testing. For instance, testing the performance should not be as simple as averaging the mean time to calculate the flightpaths for a synthesised expected amount of orders for a day. Given more time, various performance tests should be utilise to see how well the system could handle an increased number of orders than expected. This could be in the form of stress tests, load tests and endurance tests. Together, these would verify how well the system can perform when large amounts of data need to be processed, and how if it can cope with this over a prolonged period of time.