Testing Approaches

Some testing approaches that were considered included: **black box/functional** testing(**unit**, **integration**, **system**), **white box/structural** testing(**unit, integration**), **grey-box** testing, **analysis**(**manual inspection**/**static** testing) and **automated** testing.

1. **Black box testing** is mostly concerned with the **verifying the specification** (functional requirements) rather than the code itself. It focuses on the result of the output from the input without knowing the internals like how an end user may see the result. This may include:

* **Unit tests** (where assertEquals() for junit could be used to compare LngLat method output with expected output).
* **Integration testing** (to see if the using modules together give the correct result).
* **System testing** (can be for performance or to check if output files are correct with geojson).

1. **White box testing** is concerned with **structural testing**, this is when the developer know the layout and contents of the code. It usually **complements** the black box testing from above. We are concerned if all part of the code is ran and used, and generate specific tests after implementation of the code.
2. **Grey box testing** is a combination of both white and black box testing. Some testing techniques may include **regression testing**, where there are tests in place to check for when code is updated. It has advantages of both sides, where testing happens through **both the developer and the user’s interactions** with the system. With regression testing for our project, stubs are generated for tests such as the ones for the order validation where other methods were not yet available for integration testing.
3. **Analysis** can be done at any part of the development process, this may include **manual inspection techniques** and is especially good for the early stages of development. It also includes **automated analysis** like those provided by our IDE Intelij where possible problems are pointed out early. May also be known as **static testing**, because no code is dynamically ran in order for testing to happen.
4. The last approach is the **continuous feedback** and **automated testing** approach from DevOps. A **pipeline** can be set up to carry out automated testing, where new code will allow testing to happen automatically and any faults can be found immediately and notify the developer, **including build, code quality, unit tests** etc... This is very useful for any large projects with several developers working together, however it may not be as needed but still nonetheless useful for our project.

Testing Approach Justifications

For our project , we have chosen **1 to 4** from the page above as our testing approaches, a mix of **functional**, **structural**, **regression** testing(white, black and grey box testing) with manual and automated analysis (**static testing**). While testing and developing the software we will need to take some time to consider the quality process: **Completeness, timeliness** and **cost effectiveness**. Testing quality may not be of the highest standard due to the constraints of time and effectiveness. We have tried to provide a good balance of efficiency of tests vs time and prioritised completeness overall.

**Completeness** considers activities that detect important class faults. **White and black box** testing for **unit**, **integration** and **system** requirements allows us to detect such faults **dynamically**. **Manual inspection with automated warnings** from IntelliJ allows us to detect class faults before it even has started to run.

**Timeliness** considers the amount of time we have to implement testing approaches. Unit testing with Junit may use up a lot of time, but was considered important to ensure **completeness**. They serve as a building block for integration and system testing that will **verify the requirements from the specifications provided in** Project Requirements document. **Manual inspection** and catching faults early using **static testing** will help save time finding simple faults before the other tests do later on. However, this means that other testing approaches had to be dismissed, such as setting up the complete automated pipeline for continuous integration. A balance had to be considered where available resources would go to, where **CI was dismissed** as the project is itself is quite small and CI would save much more time on larger scale projects with multiple developers.

**Cost Effectiveness** considers the overall **effectiveness/efficiency** of test approaches. **Regression (grey box testing)** was deemed efficient and effective enough to be implemented in some parts of the testing approach. This allowed the testing of the validity of the orders with set parameters from the developer that could check for correct implementation of order validation methods even with updated code to meet the order validation functional requirements. **Manual inspection/static testing** is continuous and is good for any part of the development stage, but especially useful for early stages of the project where unit tests may no have been implemented yet. It is a simple way of testing and yet can be very effective.

The data to be performance tested on is very large, each day may contain about 48 order details. The specification states a system level requirement that the files will have to generated **within 60 seconds**, we have shown that to be the case. In the real world, the data is realistic, and so we can be sure that the runtime will not be any longer than that.

All tests carried out were used to verify the functional requirements for our project, a few are given as examples in the Project Requirements document.