

Software Testing 2024/2025 Portfolio - Test Results

Testing techniques used

As discussed in *TestPlan.pdf*, the testing for the system makes use of a range of techniques. The tests follow said plan and cover the requirements that have been mentioned throughout. With a good implementation of a solid test plan, the core functionality described by the ILP specification requirements can be developed effectively. There is a clear use of repeated partition-based testing for unit-level requirements with randomly generated mock data. There is a focus on stress-testing and properly testing and handling boundary/generally erroneous cases in all areas, both in the unit-level partition testing and in the full system testing when verifying end-to-end functionality. All in all, the approach clearly aligns with the design of specification-based testing and therefore will confirm the requirements are met and the intended functionality of the application is produced.

Evaluation Criteria

Measuring coverage is a very decent proxy for measuring adequacy and will very much help to show the thoroughness of a test suite and highlight major inadequacies. As my test plan and therefore my system implementation focuses very much on the error and exception handling, reaching all the extremes conditions through branches of testing will be very effective in confirming the requirements are met. The test suite can be split into the bulk of the unit tests looking specifically at the 3 core functionalities of the system (as described in *TestPlan.pdf*) and the full program system-level run-through tests.

As the majority of the unit tests use an equivalence partition-based testing design, the main evaluation criteria will look at partition coverage: this will be calculated through finding the percentage of partitions covered by executed test cases (relative to the number of total partitions identified). For these unit tests, there will also be a secondary evaluation criteria which looks at the class, method and line coverage of the 3 Java files handling each functionality. This secondary coverage can be gathered from a coverage report generated when running the test suite in IntelliJ.

For the repeated system-level tests, the focus is on the functionalities communicate sufficiently, exceptions and warnings are made clear as and when they should be, and just the general completeness of the project. The main evaluation criteria here will look at the class, method and line coverage of all files written for this project, specifically the class, method and line coverage. Again, this coverage data will be gathered from a coverage report generated when running the test suite in IntelliJ.

Results of Testing

All tests created for this Portfolio can be found in the following files:

- For the unit-level tests ensuring the core functionalities as described in *TestPlan.pdf*:
 - *InputHandlerTests.java* and *Retriever.java* for the argument validation functionality
 - *LngLatHandlingTests.java* for the drone movement system functionality
 - *OrderValidatorTests.java* for the order validation functionality
- For the system-level tests ensuring end-to-end functionality:
 - *AppTest.java*

Unit-level tests

Requirement	Partitions Found	Partitions Tested For	Partition Coverage (%)
Argument Validation	9	9	100%
Drone Movement System	11	11	100%
Order Validation	21	21	100%

Table 1: Partition coverage of unit-level requirements

Requirement	Requirement Handler	Class cov.	Method cov.	Line cov.
Argument Validation	<i>InputHandlerTests.java</i> <i>Retriever.java</i>	6/6 (100%)	10/10 (100%)	46/46 (100%)
Drone Movement System	<i>LngLatHandlingTests.java</i>	1/1 (100%)	5/5 (100%)	12/12 (100%)
Order Validation	<i>OrderValidatorTests.java</i>	1/1 (100%)	1/1 (100%)	47/47 (100%)

Table 2: Secondary coverage of unit-level requirements

It is clear from both the primary partition coverage and the secondary line coverage that these 3 functionalities are tested rigorously as the coverage is maximised.

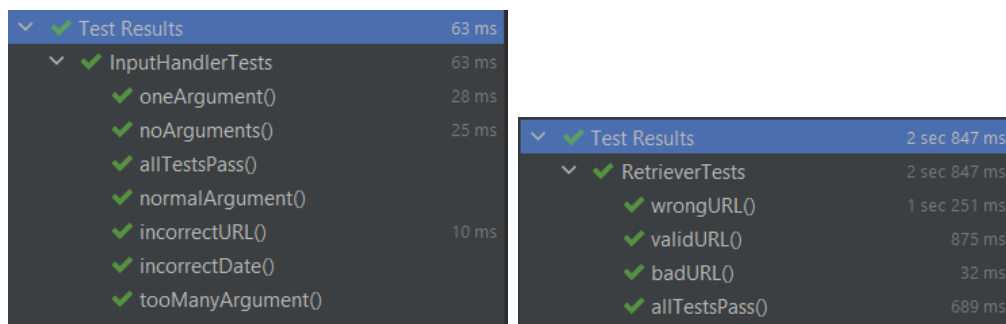


Figure 1: All tests for input validation

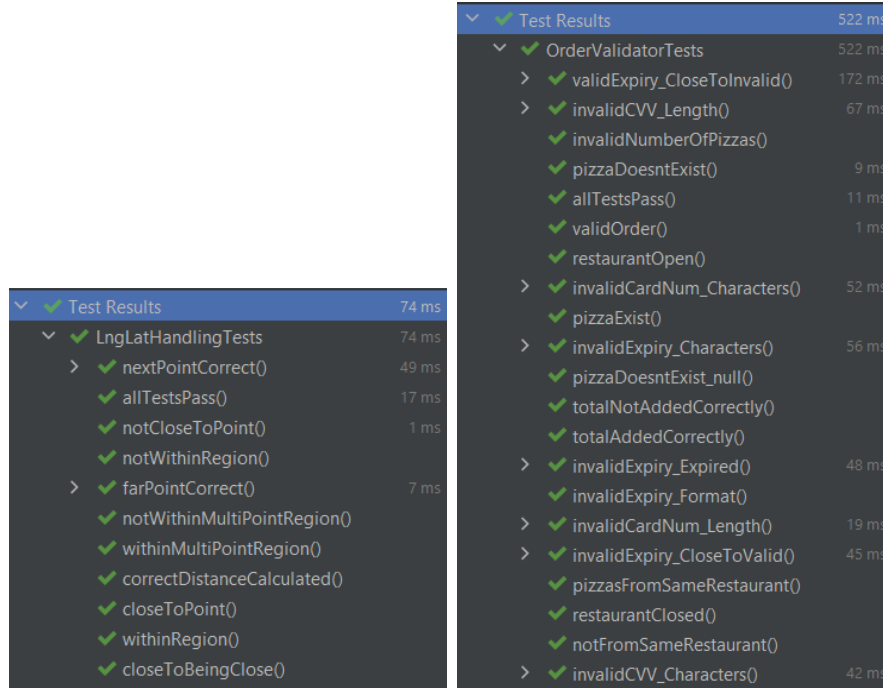


Figure 2: All tests pass for movement system functionality and order validation

System-level tests

Element	Class cov.	Method cov.	Line cov.
Input/Output	7/7 (100%)	14/14 (100%)	59/59 (100%)
Pathing	6/6 (100%)	35/35 (100%)	173/173 (100%)
Validation	3/3 (100%)	9/9 (100%)	67/75 (89%)
Main App	1/1 (100%)	3/3 (100%)	56/56 (100%)

Table 3: System-level coverage of all project files

It is clear from both this coverage that the overall functionality is tested adequately as the coverage is mostly maximised.

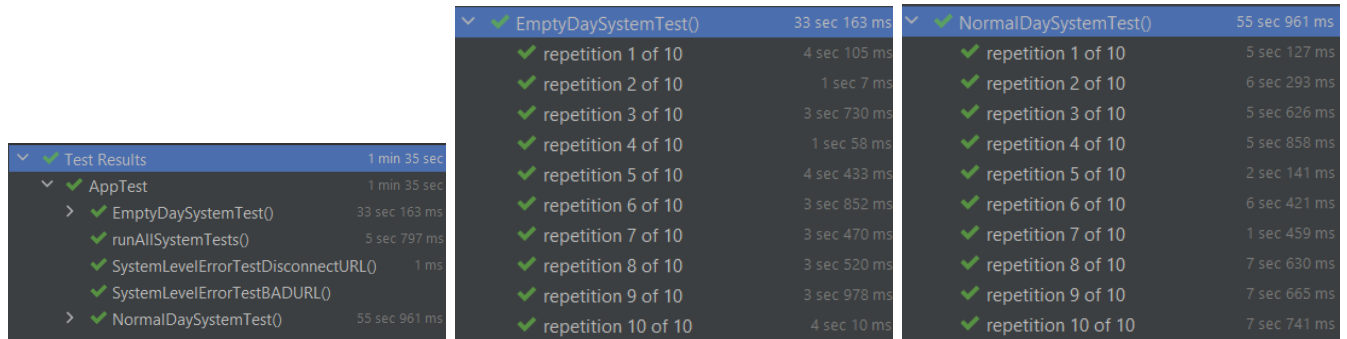
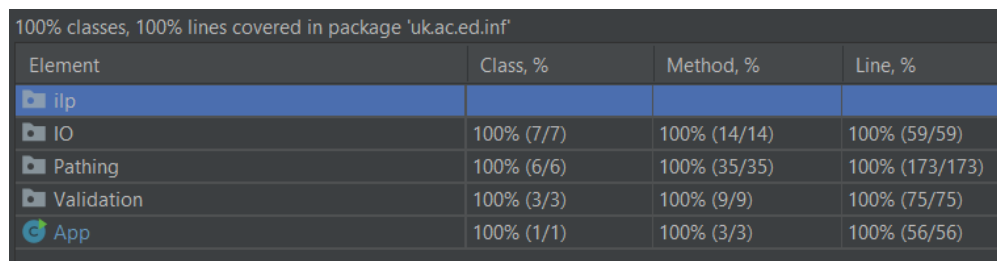


Figure 3: All tests pass for full system functionality

It is important to note that the Validation element does not achieve a 100% line coverage. This is due to the fact that system-level functionality relies on receiving data from the REST server (including order data) and all orders, as time of writing, will be validated correctly and so the validator handlers will not look at all conditions. However, as we have performed significant unit testing for this, we can see that a full coverage report generated from running all unit and system tests achieves 100% coverage in all classes, methods and lines of the project:



100% classes, 100% lines covered in package 'uk.ac.ed.inf'			
Element	Class, %	Method, %	Line, %
ilp			
IO	100% (7/7)	100% (14/14)	100% (59/59)
Pathing	100% (6/6)	100% (35/35)	100% (173/173)
Validation	100% (3/3)	100% (9/9)	100% (75/75)
App	100% (1/1)	100% (3/3)	100% (56/56)

Figure 4: Full coverage after unit-level and system-level tests ran

Evaluation of Testing

Looking at both the unit-level and system-level tests that have been carried out, the high coverage in all levels of testing (ultimately 100% as seen in Table 1, 2, 3, and 4) suggests a clear success for the testing suite. It verifies that the core functionalities have been developed adequately and that the overall system functionality meets the defined requirements. With the testing suite having specific focus on error management, having a high coverage also provides reasonable confidence in resilience to errors as it minimises untested execution paths.

It can also be clearly seen that all tests do pass (see Figures 1, 2, and 3) and that the time it took for each full run-through of the application was well under the limit of 60 seconds (a maximum of just over 7 seconds is recorded in a normal run-through repetition as seen in Figure 3). These tests have been set to fail if the runtime exceeded the limit.

There are, of course, some omissions and potential improvements as with any test suite however the tests and their impressive coverage do meet all objectives set out in when the specification requirements were analysed (in *Requirements.pdf* and again in *TestPlan.pdf*). A clear improvement for future development and testing would be to look at other criteria outside of coverage. Coverage is, after all, only a proxy for thoroughness and it may ultimately introduce a false sense of security: a test suite may look at all functionality but not necessarily all known vulnerabilities. This idea that improving coverage is not the same as improving a test suite is important to take into consideration for future work on this project and other projects.