

LO3: Testing Techniques & Evaluation Criteria for PizzaDronz Project

Introduction

This document describes what techniques were used in the testing process. We will also discuss the criteria for evaluation of the testing quality and conclude how well the tests compare to those outlined in the *LO2-Test_Planning_Document.pdf*. The tests will be evaluated based on the specified criteria, and the corresponding impact on the satisfaction of the selected requirements will be discussed.

The Tests

The main tool which was used to test the application was a Java unit testing framework JUnit. It was used both for unit and system testing, precisely as it was planned in LO2. The tests are spread across quite a few Java classes, especially when it comes to robustness testing (R4¹). The following classes contain the tests corresponding to our selection of requirements:

- *AppTest.java*
 - Unit tests for argument validation (R1).
 - System tests for program robustness that cover almost the entire application (R4). In particular, they ensure a program does not unexpectedly crash with an unhandled exception of any kind.
 - *OrderValidatorTest.java*
 - Unit tests for order validation (R2).
 - System tests for order validation (R2).
 - System tests for order association with an outcome (R3).
 - *OrdersControllerTest.java*
 - *MovementControllerTest.java*
 - *DroneControllerTest.java*
- } robustness tests (R4): proper exception handling.

As it was planned in LO2, quite a high degree of scaffolding was necessary to properly test the chosen requirements (mostly it was mocking of orders and restaurant data).

Range of Techniques

1. Functional Testing (systematic)

- The test cases were developed based on the requirements and the test plan. On the unit level, methods were verified to correctly check certain constraints, as mentioned in *The Tests* section above. On the system level, certain components were tested to have an expected correct behaviour.

2. Structural Testing

- This was a complementary procedure to functional testing. It ensured all components of the functionality being tested are covered and checked for correct behaviour. This approach has greatly helped in capturing specific test cases with a different or unusual behaviour (which obviously require special attention). For example, in identifying all possible reasons for an invalid card number, there were a lot unusual test cases. Therefore, all possible branch conditions have been checked (condition testing).

¹ R1, R2, R3 and R4 refer to the requirements chosen for testing. They are specified in *LO1-Requirements.pdf*.

3. Combinatorial Testing

- This approach was extensively used to generate inputs of different kinds in different combinations. This has helped to detect any combinations of inputs that require unusual handling. The technique was mostly used to satisfy functional requirements (R1, R2 and R3), e.g. all possible combinations of arguments (and their types) passed to the application. However it was also used for robustness testing (R4) – testing with different combinations of missing server data.

4. Model-Based Testing (not actually used)

- This technique was also considered to be applied as it has an advantage of testing certain models as separate parts of the entire system. It could have been efficiently applied because, in our application, there are two models (Movement and Orders) that can be separately tested by abstract tests, and then the entire system by executable tests. However, due to a very limited time-resource, this has not been implemented.

Evaluation Criteria for the Adequacy of Testing

1. Code Coverage (for structural testing)

This is an important metric as it indicates how much code has actually been processed by the tests. The main principle here is that a faulty statement cannot be discovered unless it is executed. It includes control and data flow, with different paths considered. Therefore, this criteria gives a fair evaluation of the testing adequacy.

This metric should be used in two ways:

- Overall test coverage – applies to the entire application (might not be representative enough for our subset of requirements, but good measure for R4 as robustness of the whole program is tested);
- Component (class) code coverage – applies to the satisfaction of R1, R2 and R3.

2. Test Passing (for functional testing)

This is quite an obvious yet the most significant criterion. The application passing the developed tests means it behaves correctly.

3. Test Diversity (for both techniques)

Only the tests that are developed rigorously, with ideally all possible types of inputs can guarantee the program's correct behaviour according to the specification provided.

Results of Testing

The application has passed all (100%) of the designed tests. This includes all unit tests (where applicable) and system tests developed for all four chosen requirements.

The testing process has helped to discover some earlier unnoticed omissions, such as:

- Checking for possible *null* in the order fields;
- Different handling of incorrect date format and dates for which there are no orders but they are valid dates. This relates to both functional correctness and to robustness.

Evaluation of the Results

1. Code Coverage:

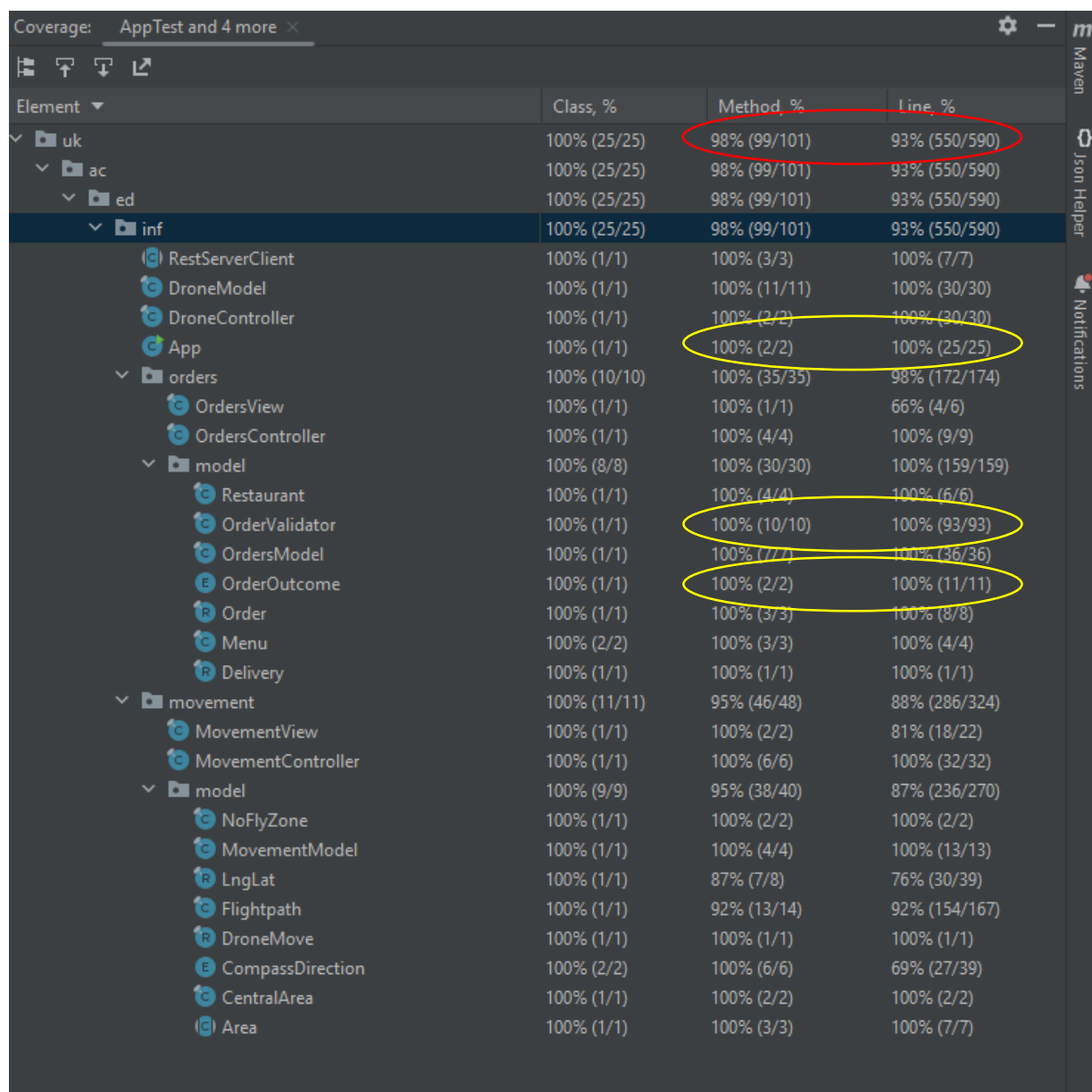
- Overall: 93% line coverage, 98% method coverage, as shown in Figure 1 (in red).
- Each component (R1 – *App.java*, R2 – *OrderValidator.java*, R3 – *OrderValidator.java* + *OrderOutcome.java*): 100% line coverage, as shown in Figure 1 (in yellow)

2. Test Passing:

- 100% tests passed, as shown in Figure 2 (in green).

3. Test Diversity:

The scenarios developed both for unit-level and system-level tests are fairly diverse as they consider all potential corner cases. In particular, all possible types and combinations of inputs, as explained in the *Range of Techniques* section. However, they are still manually generated, so there is still a possibility that some special-behaviour test cases have been omitted. This is discussed in much greater detail in the *LO4-Limitations.pdf* document.



Element	Class, %	Method, %	Line, %
uk	100% (25/25)	98% (99/101)	93% (550/590)
ac	100% (25/25)	98% (99/101)	93% (550/590)
ed	100% (25/25)	98% (99/101)	93% (550/590)
inf	100% (25/25)	98% (99/101)	93% (550/590)
RestServerClient	100% (1/1)	100% (3/3)	100% (7/7)
DroneModel	100% (1/1)	100% (11/11)	100% (30/30)
DroneController	100% (1/1)	100% (2/2)	100% (30/30)
App	100% (1/1)	100% (2/2)	100% (25/25)
orders	100% (10/10)	100% (35/35)	98% (172/174)
OrdersView	100% (1/1)	100% (1/1)	66% (4/6)
OrdersController	100% (1/1)	100% (4/4)	100% (9/9)
model	100% (8/8)	100% (30/30)	100% (159/159)
Restaurant	100% (1/1)	100% (4/4)	100% (6/6)
OrderValidator	100% (1/1)	100% (10/10)	100% (93/93)
OrdersModel	100% (1/1)	100% (7/7)	100% (36/36)
OrderOutcome	100% (1/1)	100% (2/2)	100% (11/11)
Order	100% (1/1)	100% (3/3)	100% (8/8)
Menu	100% (2/2)	100% (3/3)	100% (4/4)
Delivery	100% (1/1)	100% (1/1)	100% (1/1)
movement	100% (11/11)	95% (46/48)	88% (286/324)
MovementView	100% (1/1)	100% (2/2)	81% (18/22)
MovementController	100% (1/1)	100% (6/6)	100% (32/32)
model	100% (9/9)	95% (38/40)	87% (236/270)
NoFlyZone	100% (1/1)	100% (2/2)	100% (2/2)
MovementModel	100% (1/1)	100% (4/4)	100% (13/13)
LngLat	100% (1/1)	87% (7/8)	76% (30/39)
Flightpath	100% (1/1)	92% (13/14)	92% (154/167)
DroneMove	100% (1/1)	100% (1/1)	100% (1/1)
CompassDirection	100% (2/2)	100% (6/6)	69% (27/39)
CentralArea	100% (1/1)	100% (2/2)	100% (2/2)
Area	100% (1/1)	100% (3/3)	100% (7/7)

Figure 1 – Test coverage after executing all tests

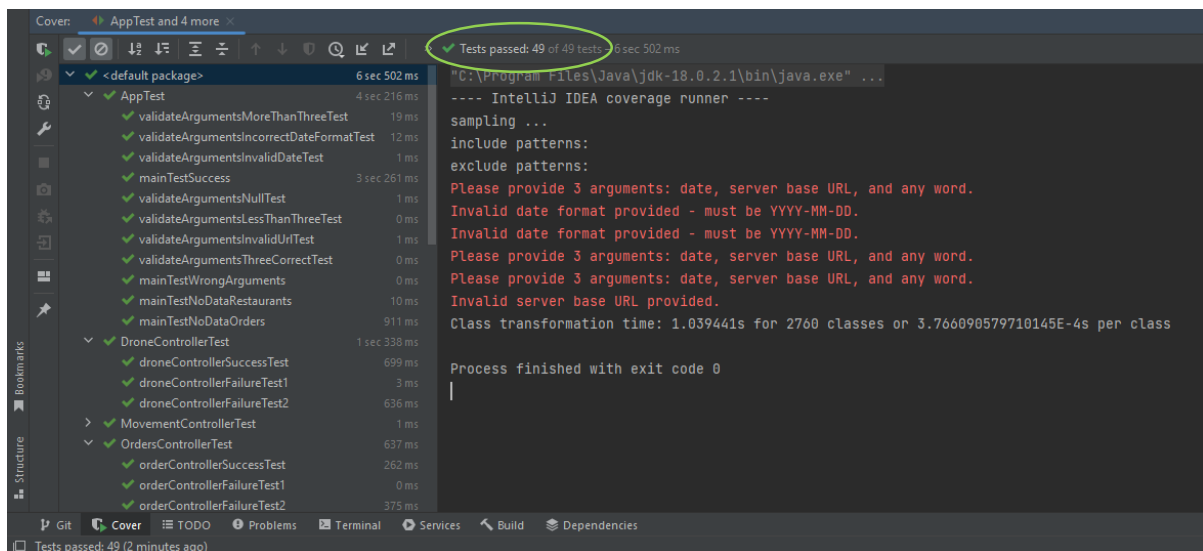


Figure 2 – Results: all tests passed

Based on the results and their evaluation, we can conclude that confidence in the testing has been improved after analysing the results according to the identified criteria. This, in turn, leads to the conclusion that the testing process has gone well and provided good results for satisfaction of the requirements R1, R2, R3 and R4.