Introduction to Software Engineering Coursework 3

Implementation

Stephen Forte - s1829305

Pierros Pantoulis - s1807852

Introduction

This document outlines the changes made on the design and requirements documents following the abstract implementation of the federalised bike rental system. The document also includes self assessment on the whole of coursework 3.

Updates to design and requirements documents

The main changes on the design and requirements documents (can all be found in those documents in green) were centered around the inclusion of the Bike class to better represent the implementation of our design, additional changes were made on the design document based on the feedback we received from Coursework 2.

We decided to include the Bike class, as it is explicitly required in the Coursework 3 instructions and we soon found out that it facilitated a number of processes in our system. We also decided to remove the BikeProviderEmployee class and add its functionality to the BikeProvider class, which made our design simpler and easier to follow.

Based on these changes, as well as some minor additions to certain classes during the implementation stage, we had to completely overhaul our Detailed Class UML diagram.

We encountered issues with the implementation of the inheritance between Quote and BookedQuote, and therefore decided to remove it, which made our code more streamlined.

Further changes on the design document included additions to the High Level Interaction, Modified Design and Get Quotes Sequence Diagrams, which were done either following feedback from Coursework 2 or during Coursework 3 to reflect the implementation.

No changes were made on the Requirements Engineering document as one of our goals throughout the implementation stage was to conform to those requirements rather than find which ones we could meet and change the rest.

Self Assessment

1. Extension submodules - 8%

a. *Implementation of extension submodule - 8%*
We fully implemented the extension submodule for discounted pricing policies using the provided interface. We designed and developed this submodule in accordance with our system as a whole so that it can be swapped out with our simple pricing module easily. We have written unit tests for the submodule checking that all functionality is correct and all tests pass.

2. Tests - 25%

a. *System tests covering key use cases - 16%*
We have written automated tests that cover all of the functionality of the specified use cases. These are concise, meaningful and simulate the real world usage of the system. All such tests have been run and pass.

b. *Unit tests for Location and DateRange - 4%*
We have written comprehensive unit tests for all functionality of Location and DateRange using varied data. All of the functionality behaves as expected and all the tests pass

c. *Systems test including implemented extension to pricing/valuation - 4%*
We have written a test that tests that the submodule can be slotted into the main system and functions correctly in it, the pricing module giving the correct price for a quote.

d. *Mock and test pricing/valuation behaviour given other extension (challenging) - 1%*
Our system is written in such a way that the other extension module can be used within it easily and correctly.

3. Code - 37%

a. *Integration with pricing and valuation policies - 7%*
Our system correctly interfaces with the pricing and valuation policies and. As evidenced by our system tests, correctly implements the default pricing/valuation behaviour.

b. *Functionality and correctness - 22%*
Our code fully implements each use case required, at points even providing additional functionality to make our model more realistic and show

connections with the non implemented use cases. Correctness of our code and the extent of functionality across the required use case is evidenced by our system tests.

c. *Quality of design and implementation - 5%*
   Our implementation closely follows our design, detailed in coursework 2, and is of high quality. Furthermore assertions are made wherever needed in order to clear up ambiguities such as the ones detailed in CW2.

d. *Readability - 3%*
   Our code is clear and easy to read, following coding conventions such as maximum number of characters per line. Furthermore, the code is well commented, including extensive JavaDoc comments for more than just the two classes required.

4. Report - 10%

a. *Revisions to design - 5%*
   A paragraph on revision to design was included, giving a high level summary of the changes made as well as the reason these changes were made.

b. *Self–assessment - 5%*
   Reflective self assessment was attempted, with a predicted grade and justification added to every area assessed.