SWEN301-P3 connolchri 300372243 Christy Connolly

Kelburn Postal Service (KPS) – Design, Quality and Behaviour Analysis

The software that has been presented to us, KPSmart is a business orientated mail management software tool used by KPS to manage and track their mail, and their interactions with transport companies that deliver their mail. This report will cover the software system in question, KPSmart, with regards to its current capabilities, the current design, quality and behaviour. The task given to us was to test the KPSmart application against the original set of specifications[1] to see how well resulting product adhered to that specification. This report will also document some of the interesting parts of that specification, and how well KPSmart did in fact, adhere to that specification.

The design of the KPS application is built upon a client-server architecture. The client-side view is a Java based GUI that interacts with the server. Each client sends packets of information to query or update the server and the server can respond or act based on the messages passed from the client. This design is well implemented, handles concurrency and message passing between server/client well. This is pretty much where the good design and quality of the system breaks down.

The server initially reads a log file on start-up and converts all the data in POJOs (plain old java objects) and loads them into the domain model. When we investigate the way the server logs data (via XML log files stored in the file system), its apparent that this wasn't an ideal design decision. File Systems are inherently good storage options for sets of unrelated data. The problem with this is KPSmart (as a mailing system) has lots of related sets of data. We have sets of transport companies, we have sets of destinations, we have methods of transport we can send by and more. A much better design choice that would've improved the design and quality of the system would've been to have used a database management system (DBMS). Fundamentally at this low level, the design decision was poor.

The domain model encapsulates the main actionable model of the tracking and management system, that is, everything that the server can directly control like the 'TransportMap', 'TransportRoute' and more. The design of the domain model is okay, but the implementation is less so, to the degree that it seems unfinished and buggy. The most complicated construct, the 'TransportMap', seems to be designed okay, it uses inheritance and method overloading well to capitalise on different forms of server updates (CustomerCostUpdate, TransportCostUpdate, TransportDiscontinue). However, there are a lot of other problems and its behaviour which hinders the overall quality of the KPS system.

Beyond the 'XMLUtil' constructs, there is a severe lack of any extensive or correct documentation. This makes it very difficult both as a tester and developer to understand the program in its essence. For instance in TransportMap both getCustomerPrice() and getTransportPrice() say they return price for customer but getTransportPrice says it calculates the price that the transport firms will charge.

Its edge case handling isn't great. For instance, if you create mail with a weight of over 200kg but a volume under 200L then it accepts it and finds a route. Similarly, if you do the converse it accepts it too. The domain only fails when both weight and volume are over. Even when it fails, its error handling and behaviour could be improved. For example, it returns a

[1] http://ecs.victoria.ac.nz/foswiki/pub/Courses/SWEN301_2018T1/ProjectThree/KPS-2017.pdf

RouteNotFound exception which isn't descriptive, it should throw something more descriptive like a MailSizeException instead.
Consistency with error handling is also a problem. When we try to ship from an origin and the origin is unknown, it throws a NullPointerException, but if we try to ship to a destination and the destination is unknown it throws a RouteNoteFound exception. This should be made into a RouteOriginNotFound and RouteDestinationNotFound exception. There's a lot of duplicated code and redundant declarations (IntelliJ tells me 281 warnings). The concept of 'Days' in MailDelivery never seems to be used in the system but are present in the log files. removeParallelEdges() has commented out code and debugging statements still, presenting its unfinished nature.
To list some functional problems, the method getRoutes in TransportMap will always return null making it useless. The calculateRoute() method in TransportMap needs a lot of work to functionally work, not even to adhere to specifications. Its outputs were non-determistic for one – its behaviour, inconsistent. For example, given a Mail object, it would often disregard required MailPriority and return different TransportRoutes containing different RouteTypes. It also threw NullPointer exceptions a lot.
The fact that the purpose of this project wasn't to unit test the program but to write BDD scenarios and I found so many problems indicates that the quality is poor.

Kelburn Postal Service (KPS) – Specification Testing Notes

The purpose of this project was to test the project against the specification not to find bugs. This was intrinsically difficult when the project itself didn't functionally work in a lot of ways. My procedure for testing was first done by breaking the specification up into requirements and restrictions. I then created steps and step definitions to prove that the domain adhered to the specification that the scenario represented. Therefore, we had a clear link between passing tests and adhering to specifications. These are located in /src/test/resources/kps and /src/test/java/kps/stepdefinitions
Whilst this was a good idea this didn't mean there weren't some interesting problems with it. There were a lot of contradictions in the specification that made it very hard to test the application. For instance, it stated that, 'KPS charges customers for this service based on the weight, volume, origin, destination, and priority' and 'The price customers are charged is based on priority, volume, weight, and destination'. The first says the origin is involved in the price calculation, the second says it isn't. There were a few instances of contradictions, I wrote a fair bit of documentation in each scenario header to explain my rationale for my test that handled these contradictions. Another problem was the ambiguity of the specification. For instance, it said 'Domestic air priority and domestic standard priority are the same'. What does this mean? Is this same in terms of charging the customer? If so then this directly contradicts 'The higher the priority, the more expensive it is for the customer' as domestic air priority is higher than domestic standard priority. Continuing with this, it said 'KPS does not accept international air priority mail for destinations if there is no transport route to that destination solely consisting of air freight once that mail has left New Zealand'. This makes sense, but its ambiguous because it doesn't tell you what the mail can do inside New Zealand, can it travel by any method (air/land/sea) or only air? It's these small details that left a lot of gaps in the specification. This meant often I had to write specification tests for all possibilities, where if one failed, the other one would pass – this is a stupid system, but to test it properly, it was my only option as the implementation didn't provide any pointers with its inconsistent and unfinished nature.