Reflections: Developing an e-store

For the mid module and final assignment, I was tasked with designing and implementing an e-store using Python and the skills developed throughout the module.

My application uses a command line interface for user input. However, assessing the final project, I now see that the same business layer of objects produced could be used to implement different forms of interface, for example, the same layer could be used in a RESTFUL(Gillis, 2020) interface.

As we learnt during initial modules the Software development life cycle (SDLC) is a standardized set of processes for building a software system. The SDLC can be used to help produce higher quality software, faster. Because of the scale of this project, I believe the ramifications of mistakes during the process are reduced. This is because a single 'small' code can be easier to modify.

SDLC Mapping

I will now consider how my implementation approach relates to the Software development life-cycle. Although I did not strictly follow the SDLC during this project I did complete activity related to its various stages throughout development. It is interesting to see how the work I did maps to the SDLC.

Planning - Reading the brief. During this phase, I developed an understanding of the system.

Analysis - From the brief, I extracted all the requirements for the system.

Design - I created class activity and state diagrams to detail the system.

Development - I implemented aspects of my initial designs during the design phase to test them out. Once the design was complete, I implemented it fully in Python.

Testing - I both 'informally' tested the system during development with temporary test scripts and later produced more detailed tests.

Deployment - I developed my system locally and copied the files across to Codio. This mimicked 'deployment'. I had some minor issues to resolve after deployment.

Maintenance - I have not completed any tasks for this phase.

Object orientated analysis was used during the analysis phase and Object orientated design was used during the design phase.

Thoughts on developing the system using Python

I enjoyed developing the project using python and have considered some of the strengths and weaknesses of the language.

I found Python very fast for prototyping. This enabled me to create quick prototypes during the design stage to test ideas out. I found Python also very fast for adding new features to the program.

I found making changes in Python that affected already existing code to be cumbersome, however. This I believe is largely down to a lack of strict class typing. In other languages if I change functions signature a compile error may be produced, in Python however I may not realize I have missed updating a function call until the program has been run.

I also found the Pythons import mechanism to be cumbersome. For starters, there are many different ways to import code. The sheer number of approaches made it harder to find reference material suitable for my needs on the internet. I also found that the need to specify not only a modules name but location strange coming from a C# background. Needing to specify locations for the imported modules meant re-structuring my project caused errors that would not have occurred in C#.

I also found the number of different Python versions to be a bit of a nuisance. For example, when I copied code from my computer to the Codio platform I had errors where the code wouldn't work because I had used a different Python version on my PC. This was not a big deal for me as I could make sure to use the same version on both my PC and Codio but I did find some examples on the internet that were not helpful due to version compatibility. I have read that on larger Python projects dynamic typing can slow down development (Markov, 2019).

I developed the Python code in 3 layers, with each layer being implemented one at time. This approach was an example of Iterative development.

Firstly, I created the Business objects for the system. I wrote simple tests scripts to validate their correct operation. Second, I wrote the interface layer. The interface layer used the

business objects to manipulate the system state. Finally, I wrote the database layer. Using this approach, I had confidence that the system operated correctly after each iteration because each layer had its own tests.

When I came to the database layers' development, I already had tests for the business and interface layer that I could use for verification.

A downside to this approach is that if I wanted to change something that was implemented in the first layer/iteration it would require changes to work done in each of the iterations that followed. An alternative approach could have been to implement all three layers at the same time. This way if I found that work in one layer required changes in another, I could address it quickly.

I think that the iterative approach was successful as it helped to produce code with a nice separation of layers.

Database

I chose to implement a database for this project. This was outside of the scope of the project but I thought it would be a good opportunity to try out some of the techniques learnt on the course.

I found the database design stage to be pretty straight forward. It was interesting to find that some of the class entities in my design did not need their own tables as the information for these classes was already captured in other tables.

I normalized most of the tables, which I found added a little extra time to the upfront development time. I imagine this would be worth it if the project ever had to actually store lots of data and needed maintenance, as it is the effects are minimal.

Database technology selection

During this module, I learnt about other forms of databases. I believe a relational database to be most suitable for this project for two reasons.

1) the project contains the highly structured data.

2) There is little chance that large schema changes will be needed so the rigidity of the database is actually a benefit (because of checks).

Thoughts on further work

If this project were to be continued, I can see how a different type of database could be useful for additional features. For example, a graph database could be used if the store

wanted to provide product recommendations. Graph databases excel with highly connected entities that have 'fuzzy' relationships and as such would suit a product recommendation system based on customers previous orders (Ontotext, 2020). A database like this could be used to supplement the existing SQL database and as such, the project could have 'the best of both worlds.

References

Gillis, A. (2020) REST API (RESTful API). Available from: https://searchapparchitecture.techtarget.com/definition/RESTful-API [Accessed 11th April 2021].

Markov, I. (2019) Why does everyone say that Python isn't good for large scale application. Available from: https://www.quora.com/Why-does-everyone-say-that-Python-isn%E2%80%99t-good-for-large-scale-applications-when-in-theory-all-languages-have-the-same-potential [Accessed 11th April 2021].

Ontotext. (2020) What is a NoSQL Graph Database?. Available from:https://www.ontotext.com/knowledgehub/fundamentals/nosql-graph-database/ [Accessed 11th April 2021].