Object-Orientated Information Systems

Module Reflection

Building an E-Portfolio

The first hurdle I encountered was building an e-portfolio. The process was arduous and filled with resistance: I hold that an e-portfolio brings no. But I understand it is a requirement of Higher Education and other institutions (Jenson & Treuer, 2014) to show a form of learning.

Regardless, I thoroughly enjoyed *building* the e-portfolio since I gained further experience in CSS, HTML structure, JavaScript and Angular. I even picked up new knowledge of Web Components and tied their similarity to frameworks like Angular, React and Vue. The experience of *building* the e-portfolio leaves me with a more profound appreciation of CSS and JavaScript; they genuinely are impressive frameworks.

The starting point for the e-portfolio was a template or two. Initially, establishing the structure of an e-portfolio was very difficult—I did not know *what* was required or *why*. I bounced hither and thither between plain HTML files, then Angular and React projects until settling on the most straightforward approach: JavaScript, HTML and CSS. And I find the simplest solution is the best one.

Object-Orientated Design (OOD)

I cannot envision designing software without the notion of “objects”. I deem single responsibility the most important from the “SOLID” principle (Single responsibility; Open-Close; Liskov inversion; Interface segregation; Dependency inversion). It is easier to build small, contained units of behaviour than to mix complex behaviours into a single whole. Nature shows biological systems that comprise several inter-connecting systems, where each contributes towards a single (complex) organism.

So too, OOD concerns designing systems as a collection of connected and dependent objects. Each object has a single responsibility, one task to accomplish. It is essential in OOD to consider the *public* contract (definition) an object exposes because it is necessary to protect its internal data. Manipulation of internal data happens through the object's contract. Sometimes, objects cannot exist without the presence of another; this is *composition.* Sometimes an object merely uses another; this is *dependency*. Whatever the relationship, the goal is always the same: to process data into information through a series of operations that share state. OOD is a foundational bedrock to building complex information systems.

UML

This module introduced UML designs. Once more, I thoroughly enjoyed considering and challenging my existing experience of UML. I picked up a deeper appreciation for modelling in UML, primarily through the discussions around UML and the system design (Unit 7). For UML exercises, I leveraged a tool called Sparx Enterprise Architect. The main contemplation of UML is the similarity between association classes and "gerund" tables in SQL structures and graph databases' relationship structures. All three represent the same *concept:* a link class with properties. UML is an excellent modelling tool to define both code and data modelling since UML is well-known and reasonably expressive.

NoSQL

Most of my working career used relational databases, and recently, graph databases by neo4j. Being familiar with SQL (T-SQL) and the relational database model—including normalisation—was a definite boon for units dealing with SQL (Unit 9, Unit 10).

The formative activity in Unit 7 forced me to take a step back and *consider* how I arrive at the third-normal form; data does not just normalise itself. Normalisation matches the concept of "refactoring", where common elements are grouped into a whole. This "refactoring" aims to reduce redundancy in table structures, and each stage of normalisation (1NF, 2NF, 3NF) uses different rules to determine how to "refactor" data from a table structure.

We had discussions between units 8, 9 and 10 in which we considered alternatives to the relational model. The focus was explicitly on the growing interest in NoSQL databases driven by Big Data. The consensus seemed unanimous that NoSQL is no going away and that the

future will require more professionals skilled with NoSQL. I feel the discussion and research around alternates have well prepared me to meet the challenges of NoSQL, based on the knowledge gained in this module.

Python

Before taking this module, I had brief exposure to Python and understood little of its basic syntax. Upon completing the Python-related units (Unit 4; Unit 11; Codio Exercises), I gained a more profound knowledge of Python language and its features.

The notion that *indentation* is a defining trait of the language is odd. However, I could argue that it is no different to curly braces ("{", "}") or TABs, found in other programming languages. I appreciate that Python emulates JavaScript: add new properties to an object instance without pre-defining those values. Dynamic attributes are both good and bad. I feel they are mostly bad because the code is not explicit (does not self-document) about what it expects. This is Python's duck typing system. And as Bailey (2021) defines, *duck typing* is concerned more about the presence of a method than the type itself. I did find that writing Python code focuses more on *intent* through code than class *structure*.

Oliphant (2007) states that Python has a "clean syntax". I think this is partly true because variables often have double underscores ("__") prefixed and suffixed, which makes them appear more unreadable. The idea of "clean syntax" is a preference and cannot be declared factual; after all, beauty is in the eye of the beholder. What I enjoy about the Python language is its handling of lists containing variable data types.

Python was not challenging to pick up, and I am far more confident in utilising the language than I was at the start of the module.

Last Thoughts

I added a reflective piece to each unit of the module because I assumed this was a requirement. Writing reflective essays is time-consuming, but I found *research* and *contemplation* to bring great value and insight into each unit's topic. The *practical, hands-on* aspects of the module were tremendously fruitful too. The module certainly was challenging from a time management perspective. I spent a considerable amount of time creating the e-

portfolio structure, design, and layout, designing the UML model, and implementing the system in Python.

Overall, I enjoyed the topics presented in this module: information systems, object orientation concepts, NoSQL and Python. Each unit flowed logically into the next and nicely built upon prior knowledge. I am satisfied with the quality of the outputs produced and knowledge gained.

References

Bailey, C. (2021) Duck Typing: Python Type Checking. Available from https://realpython.com/lessons/duck-typing/ [Accessed 23 Jul 2021].

Jenson, J.D. & Treuer, P. (2014) Defining the E-Portfolio: What it is and why it matters. *Change: The Magazine of Higher Learning*, 46(2):50-57.

Oliphant, T.E. (2007) Python for scientific computing. Computing in science & engineering, 9(3):10-20.