Object Oriented Programming: Reflections

Introduction

The module, developed my understanding of object oriented programming (OOP) and Unified Modelling Language (UML). Prior to the module I had no prior knowledge of either the object oriented paradigm or UML. The only coding experience had been gained through the Launching into Computer Science module. Over the course of the module my knowledge and skills have been developed in the core areas. The ePortfolio and Assignments demonstrate understanding and application of the key features of OOP:

Unit 1. Abstraction
Unit 2. Encapsulation
Unit 3. Inheritance
Unit 4. Polymorphism

Python Programming and the OOP Paradigm

Unit 1 provided an excellent starting point with an oppertunity to put the theoretical knowledge of object diagrams into practice. The dot notation used for methods gave clarity to the learning in the previous module where, for example, list and string methods were used. This clarified my understanding of the wide notation and conventions used in Python.

The flexibility of OOP was highlighted by embedding objects, such as the Point object in the Rectangle object (see ePortfolio Unit 1 – Codio Exercises). Furthermore, developing an understanding of the special methods such as initialization and string alongside the concept of operator overloading and polymorphism enhanced my understanding of the creativity of OOP. It became clearer how the paradigm is more suited to modelling real life systems than traditional programming approaches. For example, as an educational advisor, it was now obvious that leading management information systems, such as Arbor, use OOP to capture school, student and curriculum information.

The concept of inheritance is powerful. The ability to create similar but different objects has endless applications. The assignments focused on a driverless car and there are obvious examples of how cars, van or HGVs could inherit properties of a

vehicle class. Similarly, in my own professional experience, primary, secondary and 6th-Form would all be types of schools and could inherit the properties of a school object. The 'has a' and 'is a' rule for distinguishing between embedded and inherited objects is memorable and useful. For example, a 6th-Form **is a** school indicates inheritance. A 6th-Form **has a** set of students, indicates the student class would be a an embedded class. This is further illustrated in the ePortfolio inheritance exercises (see ePortfolio Unit 5).

The more advanced exercises (see ePortfolio Unit 10) highlighted the module approach to OOP in Python. This was, perhaps, the most insightful aspect of the course. The ability for programmers to work independently on a variety of modules has a clear benefit to the implementation phase of SDLC. The advantages are: that implementation can be quicker, as multiple programmers can work independently; encapsulation and the use of private variables limits the number of features accessible outside of the module; testing and debugging can be completed within modules, isolating any performance issues. Furthermore, this approach has obvious advantages for reusability as modules can be repeated imported into a number of applications. An excellent example are the math and csv Python modules that are routinely used. The modular approach also means that developments can be made while minimising the need to rewrite multiple applications.

Unified modelling language

This module introduced UML models and the software development lifecycle. Conceptually, and practically, this element is the most removed from my professional practice as a non-IT professional. However, it was clear that UML presents a more fluid approach to communicating the abstractions than traditional flow diagrams and there is a clear advantage in presenting a number of different views, so that communication can be relevant and fit for purpose according to the audience and their specific needs. This is, of course, a universal truth for good communication and not isolated to the applications of UML.

In particular, the high-level models are an excellent means of communication to capture the design phase of a project, while class diagrams are invaluable for programmers. There is a clear advantage to using UML when developing complex

systems involving multiple teams. This goes hand in hand with the ability to develop modules of work within the OOP.

A final reflection on UML is that the project lead for any piece of software should hold and own a suite of UML models that capture the views necessary for each stakeholder. It is also important the UMLs are updated in the maintenance phase of the SDLC. For example, an additional elements, attributes or methods may be added, or modified, for an object to reflect changes in the real-life system. For example, the UK Department for Education require schools to return a pupil census which documents the attributes of pupils. One attribute is gender which has been traditionally defined by a data set from the list [M, F]. This will be altered to [M, F, N] to reflect a real-world change in the understanding of gender. This, in-turn, will require the developers of school management information systems to update their student objects accordingly.

So What?

The OOP module has been valuable in itself, as all learning is, but what tangible benefits have been gained as a result? During the introduction to the postgraduate course, I wrote, ‘After completing this course, my aim will be to provide solutions in my own industry, both in terms of improving data driven strategy, and supporting schools wishing to develop their own computer science curriculum.’ (Jackson,2023). As a direct result of this unit, I have been able to complete analysis for a client (Eton College) who will be opening new colleges in 2027. My role is to complete the curriculum design which included timetable blocking, combinations of subjects and analysis of the demand in the local area. To aid this process, I have written a Python script to optimise choices of subjects while minimising the exposure to staffing cost. This uses students, subjects, classes, pathways and blocks as classes within an OOP system. Before completing this unit, the optimisation would have been attempted with spreadsheets. The OOP approach is more sustainable, adaptable and robust.

The OOP module has developed my skills and understanding as well as being able to progress the main aim for completing the postgraduate course.

References

Rumbaugh, J., Jacobson, I. & Booch, G. (2005) *The Unified Moddeling Language Reference Manual. Second Edition.* Boston: Addison-Wesley

Glinz, M. (2000) 'Problems and deficiencies of UML as a requirements specification language', *Tenth International Workshop on Software Specification and Design.* San Diego, 07 November 2000. San Diego: IEEE. 11-22.

Reddy, P. P. (2019) Driverless Car: Software Modelling and Design using Python and Tensorflow.

Zhou, Z. Q. & Sun, L. (2019) Metamorphic testing of driverless cars. Commun. ACM 62(3): 61–67. DOI: https://doi.org/10.1145/3241979.

Joque, J. (2016) The Invention of the Object: Object Orientation and the Philosophical Development of Programming Languages. *Philosophy & Technology* (29):335 -356.

Rumbaugh, J., Jacobson, I. & Booch, G. (2005) *The Unified Moddeling Language Reference Manual. Second Edition.* Boston: Addison-Wesley

Phillips D. (2018) *Python 3 Object-Oriented Programming*. Third Edition. Birmingham: Packt Publishing Ltd.

Glinz, M. (2000) 'Problems and deficiencies of UML as a requirements specification language', *Tenth International Workshop on Software Specification and Design.* San Diego, 07 November 2000. San Diego: IEEE. 11-22.

Babaei, P et al. (2023) Perception System Architecture for Self-Driving Vehicles: A Cyber- Physical Systems Framework. Available from: https://www.researchsquare.com/article/rs-3777591/v1 [Accessed 25 April 2024]