Object Oriented programming – Reflection

https://tbrays.github.io/e-Portfolio/

https://github.com/tbrays/e-Portfolio/tree/Object-Oriented-Programming---ePortfolio

This reflection looks at the Object-Oriented Programming module and evaluates the humanoid robot project. It will follow the What, So What, What Now format suggested by Rolf et al., (2001).

What?

The seminars were scheduled at 10 AM which made it hard to attend regularly. I did manage to attend one seminar that was rescheduled to a different time. The seminars were posted promptly and were informative, but the lack of attendance meant there was little discussion on the seminar tasks. The discussion forums were difficult, often felt forced and the topics did not always align with the unit's learning outcomes.

I found the concept of private and public variables interesting. Python does not enforce privacy for variables or constants. Instead, it relies on a naming convention to symbolise private and constant variables. This is different from other languages like JavaScript which allow private variables but only after an update.

Using pylint on the code was eye opening. I am used to using a single tab for indentation but pylint expects four spaces. The requirement for consistent indentation was difficult but I can see how it might be useful for education, teaching indentation. The linter also highlighted the importance of comments and pointed out the PEP-8 and PEP-257 style guides. I noticed that for simple functions the suggested comment format could make the comments longer than the code itself. I learned that module and function docstrings are helpful for external tools that generate software guides.

Python was the main language used but some exercises were in other languages like C# which I have not covered yet. I enjoyed using the Turtle graphics in Python as it reminded me of past experiences.

I found the "cruise ship" task open ended and felt the exercise could have been more specific. Some exercises in the book were unengaging and could have been explained better. The introduction of the humanoid robot scenario was interesting but did not always fit the assigned tasks.

Creating UML diagrams was challenging because I struggled to find suitable software to draw them. However, I discovered that I could use Mermaid format to write the code for the diagrams and interpret them using a website. Although limited this approach was helpful even though I had to adjust the diagrams to match UML symbols. I have not used UML diagrams before and have not found the need, this could be because I have not worked as part of a team. Fernandez-Saez et al, (2014) suggested that rough diagrams are better than detailed diagrams and found no diagrams is common practice. The tutor's explanation of UML diagrams was useful and helped me understand them better than when I used them in the previous module.

As someone without much software development experience I found design work particularly difficult. It was hard to visualise the necessary classes attributes and methods without prior experience. It often took trial and error to figure it out.

Additionally, I found that in some units the exercises and learning aims did not align. I found Unit testing interesting as it was new to me but took some time to get to used too. I worked with unittest and pytest and I appreciated how these tools helped automate tests and document them.

Though I had used OOP before I find polymorphism difficult to explain and use. Finding clear examples was challenging but the CS50 Python programming content helped clarify polymorphism for me.

So What?

I was able to complete the design and development tasks including using a menu interface for intuitive navigation. The robot scenario on a moon base made the task more engaging. The implementation used a 2D list to represent the moon base floor plan and I applied breadth-first search for navigation. While it worked for small maps,

I realised it could be improved by adding a heuristic and using A* for better performance.

I also used ANSI colour codes for the first time and added a splash screen to introduce the program. However, I found automated tests difficult to use because I usually run the program and make corrections as I developed. I now see how automated testing would be useful for larger programs in an industry environment.

Using pylint helped me understand the importance of consistent code formatting. It reinforced the need to adhere to standards like PEP-8 and PEP-257 even in small functions. This ensures code is readable and easily understood by others (Climent and Arbelaez, 2023). The experience with unit testing was beneficial as it introduced me to tools that improve code reliability.

Working with UML diagrams was both challenging and rewarding. While I struggled with finding the right tools, I learned that using Mermaid format was a valuable skill. The CS50 Python programming videos helped me understand polymorphism better with good examples, that I implemented in practice.

The difficulties I had with design work highlighted my lack of experience with OOP. The trial-and-error approach showed me that experience plays a big role in software development. I now understand that as I gain more experience in coding with OOP the design process will become easier.

What Now?

In the future I will focus on improving my understanding of OOP concepts especially polymorphism. I will also explore different tools for making the process of creating UML diagrams easier and more effective.

Given my experience with pylint and the importance of consistent code formatting, in readability and expansion, I will pay more attention to style guides like PEP-8 and PEP-257 in my future projects. I will also explore more tools for testing like pytest to automate tests and I will use them in future projects to better understand their real-world benefits.

In future projects I will make more use of OOP to gain experience in creating more complex systems, which will improve me design ability. I also plan to explore more programming languages and frameworks like C++ to broaden my knowledge and adapt to different coding environments as this will expose me to different approaches in software development. “…a professional software developer masters more than just one programming language.” (Stroustrup, 2013)

Reference List

Climent, L. and Arbelaez, A. (2023) “Automatic assessment of object oriented programming assignments with unit testing in Python and a real case assignment.” Available at: https://doi.org/10.1002/cae.22642.

Fernández-Sáez, A.M. *et al.* (2014) “Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments,” *Empirical Software Engineering* , 21, pp. 212–259. Available at: https://link-springer-com.uniessexlib.idm.oclc.org/article/10.1007/s10664-014-9354-4.

Rolfe, G., Freshwater, D. and Jasper, M. (2001) “Critical reflection in nursing and the helping professions: a user’s guide.”

Stroustrup, B. (2013) *The C++ Programming Language*. New York, NY: Addison Wesley Longman Publishing. Available at: https://chenweixiang.github.io/docs/The_C++_Programming_Language_4th_Edition_Bjarne_Stroustrup.pdf.