Summative Assessment: e-Portfolio Submission

E-Portfolio

Link to my e-Portfolio: https://gi0rgi-k.github.io/eportfolio/index.html

By clicking on “Object Oriented Programming” on the main page, you will be sent to this module’s page of my e-Portfolio:

https://gi0rgi-k.github.io/eportfolio/oop_module.html

Here, you can find a timeline of my journey in learning Object-Oriented Programming and descriptions of what I have learned in each Unit of the module. Under each Unit, you can click on “LEARN MORE,” and you will be redirected to a respective folder of my Object-Oriented Programming Github repo, where you will find all of the artifacts and tasks I have accomplished for the respective Unit.

This is a direct link to the Github repo:

https://github.com/gi0rgi-k/OOP-UoE

Professional Skills Matrix

Skills	**Skill Level Grade** *Beginning of Module*	**Skill Level Grade** *End of Module*
Time management	**4**	**4**
Critical thinking and analysis	**4.5**	**4.5**
Problem-solving	**4.5**	**4.5**
Communication and Literacy skills	**4**	**4**
IT and Digital	**4**	**4.5**
Numeracy	**4**	**4.5**
Critical Reflection	**4.5**	**4.5**
Python OOP	**1**	**4**
Software Design (UML)	**1**	**4**
Academic Writting	**4.5**	**4.5**

Skill Level Grade: 1-5

Figure 1. The increase in skill level after the OOP Module

Reflective Essay

As a Product Manager without a Computer Science education, I had to delve into various technical topics and learn them on the job to understand the complexities of the digital products I have been working on, to communicate with engineers, and to deliver products that our customers would love. I enrolled in an Object-oriented programming module primarily for two reasons. One is to deepen my knowledge and better understand the theoretical foundations and software development process. The second reason is a pure curiosity to learn something new and exciting and continue growing as a person.

In this paper, I will reflect on my learning of Object-Oriented Programming (OOP) at the University of Essex and describe how the module helped me gain new skills and improve professionally. For a consistent structure, I will follow Rolfe et al.'s (2001) approach, where I will explain essential events or topics I encountered in the module (What), interpret the events and why they were important (So what), and what I learned from them and what was the impact of the learning (Now what). In some cases, I will also use the six steps of the Gibbs Reflective Cycle to reflect on my learning progress in the module (Galli, 2022).

One core focus of the module was to understand Object Oriented approach to software development. The four core principles of OOP—abstraction, encapsulation, inheritance, and polymorphism—form its foundation, each contributing to creating modular, reusable, and maintainable software (Lott & Phillips, 2021).

Abstraction refers to only displaying parts of the functionality of a class that is relevant to a specific use case. It simplifies the system by only focusing on the task

at hand. This allows developers to focus on high-level functionalities rather than low-level implementation details (Lott & Phillips, 2021).

Encapsulation is a technique for preventing access to some of the object's components, which can avoid unintended updates in data and the code. This principle is implemented by having different access types, such as private, protected, and public, which control a class's access to attributes and methods (Noback, 2020).

Inheritance refers to the capability of a child class to inherit properties and behaviors from a parent class. This principle promotes code reusability, as standard functionalities need not be rewritten but can be inherited and extended by child classes, making developers more efficient (Lutz, 2013). It also simplifies maintenance, as engineers only need to focus on the relevant level of inheritance if an update or bug fixing is required, not the complete code for every minor update.

Polymorphism allows methods to act differently based on the object. This enhances flexibility and integration by allowing all objects from the same class to access the same interface (Åkerblom & Wrigstad, 2015). This also facilitates reusability since developers can create generalizable Classes which can be used in different contexts.

The module helped me to understand the importance of developing applications according to these principles for improved reusability, modularity, and maintenance (Padhy et al., 2018), which redefined my prior understanding of programming. As a Product Manager, when working with engineers on the release of new features, I was able to discuss more technical aspects of the application and understand their approach to writing code – why they structured different parts of the code in specific

ways, do they prioritize reusability and modularity or for which features did they have to disregard these principles due to urgent delivery, accruing technical debt and making software less scalable. Understanding these concepts during the OOP module helped me to understand the digital products I am working on at a deeper and more technical level. This module also inspired me to discuss and promote the OOP principles to my team and convince them that such a foundation is worth the initial effort to ensure maintainability and reusability that will facilitate faster new developments and bug fixes in the long run.

Learning UML and using it to plan the implementation and design of the architecture of driverless car systems has given me a new language for communicating with various stakeholders during the development process. UML offers a standardized way to visualize a system's design. It is essential for planning, documenting, and communicating complex software architecture. Use Case diagrams depict how the application would work from a user’s perspective. It shows the interactions between external entities and different aspects of the system (Rumbaugh et al., 2004). Learning this technique improved my communication with engineers as it helped me align with them on the software's business requirements and expected technical capabilities.

Class diagrams show a system's structure by showing its classes, attributes, methods, and relationships between objects (Rumbaugh et al., 2004). For example, for the driverless car assignment, the User Interface class would call methods from the Sensor and Navigation classes in the application; class diagrams depict such relationships between objects. While I would not need to use this for work, creating a

class diagram helped me better understand the concepts of classes and their relationships. Hence, it helped me to understand the OOP approach better.

Sequence diagrams focus on a system's changing behavior, while activity diagrams represent the flow of changing activities (Rumbaugh et al., 2004).

Collaborative discussions helped me gain a broader perspective on the module's topics. For example, I noticed that some of my peers had different understandings of the priorities of factors for reusability (Padhy et al., 2018), which led me to appreciate the importance of documentation and requirement analysis more than I initially thought. As a result, I rethought the priorities from the Initial Post for the final Summary Post.

The Codio exercises and the development of the driverless car application have provided me with valuable hands-on experience, contributing to my personal and professional growth. The practical exercises have given me the confidence to independently develop application ideas using the Object-Oriented Programming (OOP) approach. Moreover, the process of debugging and unit testing in Python, with Assert functions, has increased my appreciation for the work performed by the developers on my team. This experience has enhanced my technical skills and gave me a better understanding of the software development process.

References

Åkerblom, B., & Wrigstad, T. (2015) Measuring polymorphism in python programs. Proceedings of the 11th Symposium on Dynamic Languages 114–128. DOI: 10.1145/2816707.2816717

Galli, F. (2022) Gibbs' cycle review. Emotions as a part of the cycle. e-Motion Revista de Educación Motricidad e Investigación 92-101. DOI:10.33776/remo.vi19.7224

Lott, S.F. & Phillips, D. (2021) Python object-oriented programming: Build robust and maintainable object-oriented python applications and libraries. Birmingham: Packt Publishing.

Lutz, M. (2013) Learning Python: Powerful object-oriented programming.

Noback, M. (2020) Object design style guide. Manning Publications.

Padhy, N., Satapathy, S., & Singh, R.P. (2018) 'State-of-the-Art Object-Oriented Metrics and Its Reusability: A Decade Review', in: Satapathy S., Bhateja V., Das S. (eds) Smart Computing and Informatics. Smart Innovation, Systems and Technologies. 77. Springer.

Rolfe, G., Freshwater, D. & Jasper, M. (2001) Critical reflection in nursing and the helping professions: a user's guide. Basingstoke: Palgrave Macmillan.

Rumbaugh, J., Jacobson, I. & Booch, G. (2004) The Unified Modeling Language Reference Manual. 2nd ed. Addison-Wesley

Scrum (2024) Learn About the Sprint Retrospective Event. Available from: https://www.scrum.org/resources/what-is-a-sprint-retrospective [Accessed 2 June 2024].