Summative Assessment 3: e-Portfolio Submission

Object Oriented Programming

Word count: 1078

E-portfolio:

https://busilas.github.io/eportfolio/module2.html

Reflective Essay

Since my youth, I have been learning to play the piano and guitar, and I have consistently practiced for several years. Over the last 12 weeks, I have immersed myself in exploring Object-Oriented Programming (OOP) using Python. I have been developing a driverless car system program and several smaller programs. Through this experience, I discovered similarities between the process of creating software and mastering a musical instrument. Engaging with OOP through the lens of playing a guitar has been a revelatory and enlightening journey. In this reflective essay, I examine the parallels between programming in Python and playing the guitar, drawing connections between fundamental concepts in OOP and musical principles. By doing

so, I appreciate the artistry, creativity, and discipline required in both fields, as well as the importance of adhering to fundamental principles and engaging in continuous practice.

Learning to play the guitar starts with understanding its fundamental structure, which includes strings, frets, and body. Similarly, mastering OOP with Python begins by understanding what it means for a program to be object-oriented. This involves understanding the syntax used to define a Python class and how various data types are defined (Lott & Phillips, 2021). The four core features of OOP—abstraction, encapsulation, inheritance, and polymorphism—form the foundation of this programming paradigm and can be effectively compared to the various aspects of guitar playing.

Abstraction empowers developers to focus on the essential structures of a program, disregarding the intricate details, similar to how understanding music theory is crucial when playing the guitar (Lott & Phillips, 2021; Noback, 2020). Just as a guitarist understands the overall structure of a piece of music, a programmer uses abstraction to handle complexity by interacting with higher-level objects. This concept enables guitarists and programmers to simplify complex tasks, making it easier to create and maintain intricate compositions or software systems.

Encapsulation is analogous to keeping a guitar's components separate and well-organized, ensuring that each part operates independently while contributing to the overall sound. In OOP, encapsulation entails grouping related data and methods that operate on that data within a single class or unit (Lott & Phillips, 2021; Noback, 2020). This helps protect the internal state of an object from unintended interference. This principle ensures that objects are self-contained, and their functionalities are

exposed in a controlled manner, similar to how a guitarist maintains the integrity of their instrument to produce consistent sounds (Omer et al., 2023).

Inheritance in OOP is comparable to transferring techniques from one style of guitar playing to another, enabling the reuse of skills and patterns. This mechanism allows new classes to acquire attributes and behaviors from existing classes, thereby fostering code reusability and establishing a logical hierarchy. (Lott & Phillips, 2021; Noback, 2020). For guitarists, this is akin to mastering a fundamental technique, like fingerpicking, and utilizing it in different musical genres. Inheritance allows programmers to build on existing code, enhancing productivity and ensuring that established functionalities are preserved across different parts of a software system.

Polymorphism is akin to a guitarist's capacity to adjust their playing style to different genres or songs, allowing the same method to be utilized in diverse contexts. Polymorphism enables the treatment of objects as instances of their parent class rather than their specific class, thereby enhancing flexibility and integration (Lott & Phillips, 2021; Noback, 2020). This capability is akin to a guitarist's versatility, enabling them to play classical, rock, or jazz with the same fundamental techniques, thereby showcasing the adaptability and dynamic nature of both music and programming.

The Unified Modeling Language (UML) serves as a guiding framework in both programming and music, providing a standardized way to represent and communicate complex structures (Nyisztor, 2018). Just as a musician follows musical notation, I use UML diagrams to outline the blueprint of software systems before converting them into code (Rumbaugh et al., 2004). Use case diagrams and state machine diagrams in UML help identify actors, interactions, behaviors, and system dynamics, similar to planning the structure and flow of a musical performance.

The process of transforming UML models into Python code can be compared to bringing a musical score to life through performance. Detailed planning in UML played a crucial role in the implementation, ensuring that the final software closely adhered to the original design (Nyisztor, 2018). Class diagrams, a vital component of UML models, enable the establishment of a well-organized and clear code structure, mirroring the precise arrangement and sequence found in music composition.

Exploring advanced OOP concepts such as constructors, abstract classes, and interfaces has broadened my understanding of the intricacies involved in crafting well-designed software systems (Gabbrielli & Martini, 2023). Constructors and abstract classes shape the structure and behavior of objects, similar to how musicians use various techniques to craft their sound. Interfaces promote modularity in code, enabling the integration of diverse components into a cohesive whole, reflecting the harmony and coherence essential to both programming and music.

Debugging code and handling errors are similar to tuning a guitar to achieve flawless software performance. Understanding data structures in Python is akin to knowing chord progressions, whereas effective search techniques are identical to finding harmonies. Experimenting with data structures and search algorithms fosters creativity and efficiently solves problems. Thoroughly packaging and testing Python code is akin to preparing for a concert, ensuring high-quality, client-ready software (Climent & Arbelaez, 2023). Unit tests and tools validate code functionality, like a sound check before a performance. Proper code packaging and testing ensure deployment readiness and high quality.

By drawing an analogy to playing the guitar, my exploration of OOP has not only deepened my comprehension of software development, but also expanded my outlook on the inventive processes present in music and programming. The parallels between

these fields emphasize the significance of fundamental principles, dedicated practice, and the integration of individual components into a harmonious whole. Just as a masterfully performed piece of music captivates audiences, well-designed software systems can blend functionality with beauty, evoking emotions and creating lasting impacts.

In conclusion, the synergy between coding and music emphasizes the artistry, creativity, and precision needed to create exceptional work in both fields. This reflective journey deepened my appreciation for the harmonious interplay between code and music, showcasing the transformative power of creative expression in diverse forms. By embracing the principles of OOP from a guitarist's perspective, I have gained insights into the interconnectedness of seemingly disparate fields and the universal language of creativity that binds them together.

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', Computer Applications in Engineering Education, 31(5), pp. 1321–1338. doi:10.1002/cae.22642.

Gabbrielli, M. and Martini, S. (2023) 'Programming languages: Principles and paradigms', Undergraduate Topics in Computer Science [Preprint]. doi:10.1007/978-3-031-34144-1.

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

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

Nyisztor, K. (2018) UML and object-oriented design foundations: Understanding object-oriented programming and the Unified Modeling Language. Karoly Nyisztor.

Omer, S.K., Mirkhan, S.D., Hussein, N.N., Ali, A.Z., Rashid, T.A., Ali, H.M., Hamza, M.Y. & Nedunchezhian, P. (2023) 'Comparative analysis of encapsulation in Java and python: Syntax and implementation differences', The Journal of Duhok University, 26(2), pp. 379–389. doi:10.26682/csjuod.2023.26.2.35.

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