**IT5016D: Software Development Fundamentals**

**Assessment #2: Reflective Journal**

|  |  |
| --- | --- |
| **Student Name:** | **David Jacobs** |
| **Student ID:** | **20230671** |
| **Due Date:** | **21 / 4 / 2023** |

**Exploration of design patterns.**

Design patterns are proven development standards that can accelerate the development process without having to reinvent patterns every time a problem arises. Design patterns are proven solutions to solve commonly reoccurring coding issues when attempting to meet a program’s requirements. An example of a design pattern is theSingleton method which lets you ensure a class has only one instance, while providing a global access point to the instance.

**Analysis of design principles.**

Design principles, which are more generalized and abstract than design patterns, are general guidelines that can guide your class structure and relationships when coding. The most common set of design principles are known by the acronym SOLID. This set was developed by Robert C. Martin in the year 2000 and includes the following.

* **Single Responsibility Principal**

This principle is a mandate that expresses a class should have one, and only one, reason to change. Following this principle means that each class solves only one problem and every class or module only has a responsibility for one part of the software’s functionality.

* **Open / Closed Principal**

This is where you are able to extend the behaviour of a class without modifying it. This means a class is open for extension and the behaviour of the class can be extended whilst being closed for modification where the source code is set and cannot be changed.

* **Liskov Substitution Principal**

Broadly, this principle simply requires that every derived class should be substitutable for its parent class. This means that if a subclass redefines a function also present in the parent class, a client-user should not be noticing any difference in behaviour**,** and it is a substitutefor the base class.

* **Interface Segregation Principal**

This is whereusing many client specific interfaces are better than one general purpose interface. I.e. when a subclass inherits methods from a base class that it does not need.

* **Dependency inversion Principal**

This means that developers should depend on abstractions rather than implementations whereby the handling of complexity is met by hiding unnecessary information from the user. This is one of the core concepts of object-oriented programming (OOP) languages.

**Integration of research and practice.**

* In writing the code for my assignment I have used a main class which acts as the point of execution for my program. I also used a sub class that is derived from the main class with most of my functions.
* I have applied the necessary indentation to nest my code.
* I have added comments throughout my code to explain how the classes and functions operate.
* I have used global variables sparingly and rely of my classes to iterate processes within my program.

**Summary**

I have enjoyed learning to code in Python. Using classes was difficult at first but when I wrote the code for classes and saw the program functioning properly I felt I had a good understanding of how classes work. I feel confident I could write code in python again based on the software project I completed and the short programs I wrote leading up to my assignment.