Research Assignment 1: SOLID & GRASP Explained

Following proper design practices is extremely important for software engineering as it could provide a myriad of benefits for software engineers as well as the prevention of possible anti-patterns or other problems. The SOLID and GRASP principles will be further explored below as the implementation of these principles may lead to numerous benefits for software engineers and their projects.

SOLID

SOLID is an acronym that refers to a collection of design principles. These principals are the single responsibility principle, open-closed principle, Liskov substitution principle, interface segregation principle, and dependency inversion principle (Watts, 2020). These principles were initially developed to help combat the possibility of software projects becoming fragile, rigid, and immobile. Implementing these patterns allows the project to become more maintainable as you can make changes in one area of the project without greatly impacting or breaking others.

1. **Single Responsibility Principle**
   The single responsibility principle refers to each module or class only having one purpose or reason for being created (Abba, 2022). For example, instead of having a claim class containing functions for both deleting and creating claims, we would have a class for deleting claims and a separate class for creating claims. This would allow for more reusable code in the future.
2. **Open-Closed Principle**
   This principles dictates that entities should be OPEN for extension but CLOSED for modification (Abba, 2022). This means that entities such as classes should be created in a way where a different class can implement its functionality instead of modifying the original class's source code. For example, we could create an interface for calculating test scores that has a method called "calcTestScores". Other classes could then use this method and change its implementation instead of creating a different method inside of the original entity every time a new entity needs to calculate test scores.
3. **Liskov Substitution Principle**
   The Liskov substitution principle implies that when class inherits from a parent class, the child class should have a use case for all the properties and behaviors of the parent class (Abba, 2022). This means that when inheriting from a class, the child class should be able to make full use of the parent class and not only use part of the parent class. For example, if you inherit from a math class with both an adding and subtraction method, an inheriting class should be able to use both methods instead of only adding or only subtraction.
4. **Interface Segregation Principle (ISP)**
   The ISP refers to the idea that an interface should be created in a way that the user would only have access to behaviors that are related to their needs. Unlike the Liskov Substitution Principle, the ISP dictates that it is unreasonable to create an interface with a wide variety of methods as it would likely not all be relevant to every child class (Abba, 2022).
5. **Dependency Inversion Principle (DIP)**
   The Dependency Inversion Principle refers to the idea that high level modules should not import from low level modules, but instead that both should depend on abstraction. Furthermore, abstraction should not be dependent on details, details should also be dependent on abstraction (Abba, 2022). To implement this, classes should not be dependent on each other, they should instead be dependent on properties from interfaces. This helps prevent a rigid system.

GRASP

According to Craig Larman in his book Applying UML and Patterns, GRASP stands for general responsibility assignment software patterns and is a set of "nine essential principles in object design and responsibility assignment" (1997) (BOLDare, 2022).

The nine GRASP guidelines (or problems) are as follows:

- **Creator** - Who generates a new instance of a class or an object? According to the Creator principle, a certain class B should oversee producing a particular category of objects A (BOLDare, 2022).
- **Information Professional** - What obligations can be given to an object? According to this approach, you should think about giving the class that contains the operation's inputs the responsibility of performing the operation if the operation requires inputs (BOLDare, 2022).
- **Low Coupling** - What types of connections are made between objects? How do you encourage low dependency, minimal modification, and greater reuse? When two pieces of code are dependent on one another, coupling occurs between the two. Even if just because the code can no longer be understood separately, coupling adds complexity (BOLDare, 2022).
- **The controller** - which governs how domain layer requests are transferred from UI layer objects (BOLDare, 2022).
- The **High Cohesion Principle** zooms in and focuses on specific element-based roles and responsibilities. It follows an intra-element concern, by assessing how closely related the components and parts within an element correlate with one another, as well as to what degree they belong together in the same encapsulation (Grzybek, 2022).
- **Indirection** can be seen as somewhat of a countermeasure against tight coupling whereby different element parts, components or services with overlapping concern are applied, or directed to a new element. This avoids direct coupling (Grzybek, 2022).
- **Polymorphism,** a core principle within the context of Object-Oriented Programming, also has a place within GRASP. In the context of the latter, the principal points to the variations in implementation of elements and their components in accordance with the specific requirements of the use case scenario (Grzybek, 2022).
- **Pure fabrication** - this class, which domain-driven design refers to as a "service," does not reflect anything related to the problem domain and was instead developed to guarantee High Cohesion and Low Coupling (Grzybek, 2022).
- The **Protected Variation** principle can arguably be regarded as being the most important principle in GRASP. This principle greatly contributes to the dynamicity that is required by most software projects. It is primarily concerned with how easily software elements can be adapted to suit specific contexts and environments (Grzybek, 2022).

References

Abba, I. V., 2022. *SOLID Definition – the SOLID Principles of Object-Oriented Design Explained.*
[Online]
Available at: https://www.freecodecamp.org/news/solid-principles-single-responsibility-principle-explained/
[Accessed 6 October 2022].

BOLDare, 2022. *SOLID, CUPID & GRASP – three (more) principles that every developer should know about.* [Online]
Available at: https://www.boldare.com/blog/solid-cupid-grasp-principles-object-oriented-design/#what-is-solid-and-why-is-it-more-than-just-an-acronym?-what-is-grasp-and-why-is-it-challenging ?
[Accessed 6 October 2022].

Grzybek, K., 2022. *GRASP – General Responsibility Assignment Software Patterns Explained.* [Online]
Available at: http://www.kamilgrzybek.com/design/grasp-explained/
[Accessed 6 October 2022].

Watts, S., 2020. *The Importance of SOLID Design Principles.* [Online]
Available at: https://www.bmc.com/blogs/solid-design-principles/#:~:text=SOLID%20is%20an%20acronym%20that,some%20important%20benefits%20for%20developers .
[Accessed 6 October 2022].