THE SOLID PRINCIPLES

- **S**ingle Responsibility Principle
- **O**pen/Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

SINGLE RESPONSIBILITY PRINCIPLE

- The general principle of Cohesion, which says that the responsibilities of any component should be tightly aligned and focused on a single purpose.
- Reduce Complexity, even though the number of classes might increase

OPEN/CLOSED PRINCIPLE

- Software entities should be open for extension but closed to modification
- **open** if it is still available for extension
- **closed** if it is available for use by other class, and therefore should not be modified

INTERFACES

- An interface is like a base class, but only allows for method declarations

ABSTRACT AND PURE VIRTUAL CLASSES

- May include data members and some method implementations
- The modern Open/Closed Principle encourages developers to

  use interfaces, abstract classes, and pure virtual classes to declare public data members
- Derive concrete classes from these abstract components

OPEN/CLOSED PRINCIPLE

- Ways to achieve the open/closed principle

  • **Inheritance**

  • Move public methods into their own abstractions, namely interfaces, abstract classes, or pure virtual classes

  • Have concrete classes inherit from these abstraction

  • **Aggregation**

  • Encapsulate behaviors in sub-part objects and allow those sub-part object to change dynamically

• This technique has been embodied in something called the strategy pattern – more on this later

• Parameterization

• Use a generic to capture a template solution and instantiate it with the specific data types

LISKOV SUBSTITUTION PRINCIPLE

· Each class that inherits from another can be used as its parent without the need to know the differences between them.

INTERFACE SEGREGATION PRINCIPLE

· An interface is a “window” into the functionality of a component

· Clients should not be forced to depend upon interfaces that they do not use

DEPENDENCY INVERSION PRINCIPLE

· Layer the system: some levels, such as reusable libraries or frameworks, are more abstract or policy-setting, and others will be more detail-oriented.

· Abstractions should not depend on details

· High-level modules should not depend on low-level modules

· Both low-level and high-level modules should depend on abstractions

SOFTWARE ENGINEERING PRINCIPLES PAPER

Software engineers aim to build quality products on time and within budget

CORE PRINCIPLES

· Modularity

· Abstraction

· Encapsulation

COMMON PARADIGMS

· Object orientation

· Aspect orientation

· Functional programming

· Logic programming

· Genetic programming

· Structured program

CORE PROBLEM

· Programmer often don't understand the core principles, and therefore don't benefit from their guidance, especially in multi-paradigm software development

· Lack of unifying definitions hinders tools support

BEST PRACTICES, PATTERNS, AND IDIOMS

· Best practices are procedures or techniques that help developers adhere to principles,

· Patterns exemplify principles, by providing proven solutions to reoccurring problems in specific contexts.

· Idioms are techniques or solution for expressing a certain algorithm or data structure in a specific programming language

PARADIGM-INDEPENDENT DEFINITION FOR ENCAPSULATION

· Ensure that the private implementation details of a component are insulated so they cannot be accessed or modified by other components.

Practices and Criteria:

• Conceptual barriers

• Programmatic barriers

• Usage barriers

NON-REDUNDANCY AND COMPLIMENTARY –

CRITERION #1

· Modularity deals with the decomposition of system into components, whereas abstraction and encapsulation deal with individual components

· Abstraction and encapsulation might be considered duals of each other

NON-REDUNDANCY AND COMPLIMENTARY –CRITERION #2

1. A simple program snippet with good Modularity, Abstraction, Encapsulation

2. Same as #1, but with just good Modularity

3. Same as #1, but with just good Abstraction

4. Same as #1, but with just good Encapsulation