

Coding Principles

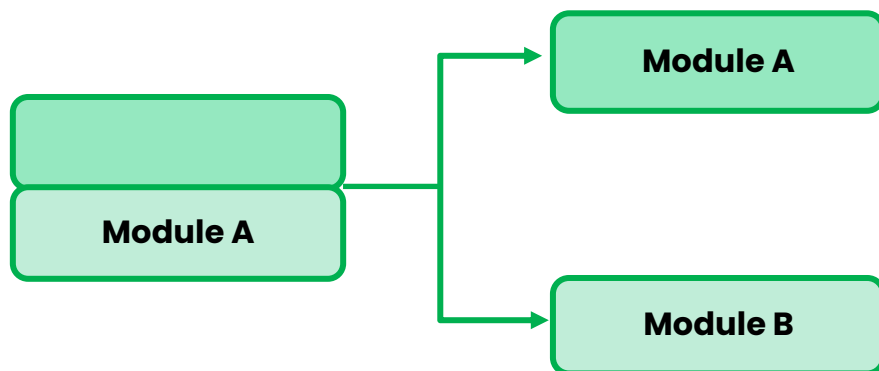
Achieving high-quality code and design is essential for building software that is reliable, scalable, and maintainable.

Below are the principles of effective software development:

1. Separation of Concerns (SoC)

This principle focuses on dividing a system into distinct sections, each responsible for a specific functionality.

For example, the Model-View-Controller (MVC) architecture in frameworks like Django (Python) and Ruby on Rails separates business logic, user interface, and data management, making the codebase easier to understand and maintain.



Alt text: Separation of concerns concept

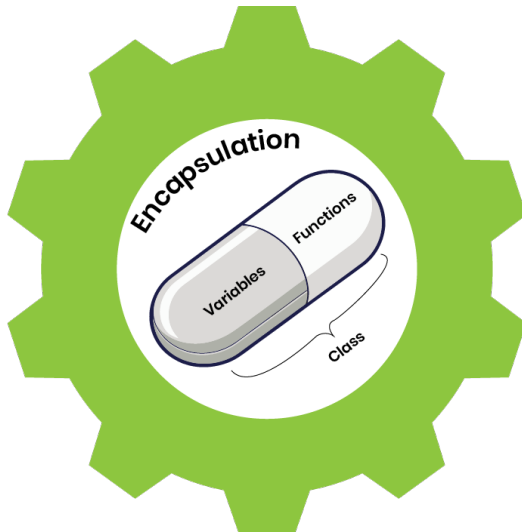
2. Single Responsibility Principle (SRP)

SRP states that a class or module should have only one responsibility or reason to change. This principle is crucial in object-oriented languages like Java and C++, where adhering to SRP ensures that each class handles a specific task, reducing complexity and making the code easier to test and maintain.



Alt text: Applications in Java and C++

3. Modularity and Encapsulation



Modularity involves breaking down a system into smaller, manageable parts, while encapsulation hides the internal details of a module. This is a key feature in languages like Python and JavaScript, where modules and classes allow developers to encapsulate data and behavior, promoting reusability and simplifying debugging.

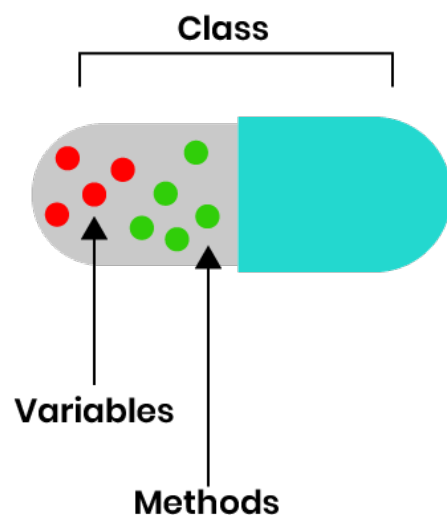
Alt text: Encapsulation concept

4. Abstraction and Information Hiding

Abstraction involves simplifying complex systems by focusing on essential details, while information hiding limits the exposure of internal workings. In languages like Java, abstraction is implemented through interfaces and abstract classes, making the code easier to manage and adapt as requirements evolve.

```
Class
{
    data members
    +
    member functions
}
```

ENCAPSULATION



Alt text: Data encapsulation

5. Don't Repeat Yourself (DRY)

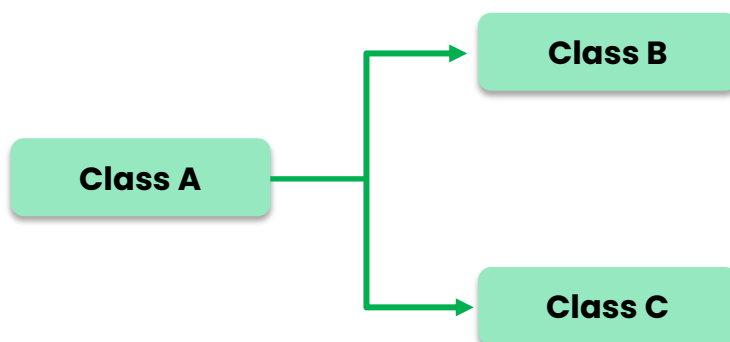
The DRY principle discourages code duplication, emphasising code reuse through abstraction and modularity. For example, using functions or methods in JavaScript or Python helps avoid redundant code, making maintenance easier and reducing the chances of introducing errors when changes are made.



Alt text: Modularity in coding

6. Open/Closed Principle (OCP)

The OCP principle advocates that software entities should be open for extension but closed for modification. In languages like Java this can be achieved using design patterns such as Strategy or Decorator, allowing new behaviors to be added without altering the existing codebase.



Alt text: Open closed class concept

7. Liskov Substitution Principle (LSP)

LSP states that objects of a parent class should be replaceable with objects of a subclass without breaking the functionality of the system. This principle is vital in languages like C++ and Java, where inheritance is commonly used. Adhering to LSP ensures code reusability and flexibility.



Alt text: Code reusability

8. Interface Segregation Principle (ISP)

ISP suggests that a client should not be forced to depend on interfaces it does not use. In languages like C# and Java, this is achieved by creating smaller, more focused interfaces, promoting better modularity and reducing the impact of changes in one part of the code on other parts.

For example, in an interface, IWorker (methods for work tasks), IHuman (methods for breaks), and IRobot (methods for maintenance), demonstrating the Interface Segregation Principle by ensuring each class has only relevant methods. By structuring the classes this way, each class implements only the methods relevant to its role, thus adhering to the principle of segregating interfaces.

Clients should only be required to interact with interfaces they actually need.

Good

Iworker
InitiateTask()
BeginTask()
BeginTask()
EndTask()

IHuman
ShortBreak()
ExtendedBreak()

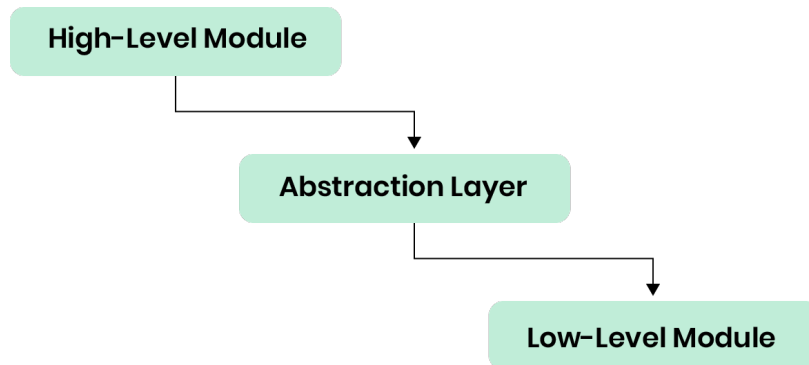
IRobot
PowerUp()
MaintenanceCheck()



Alt text: Interface segregation principle

9. Dependency Inversion Principle (DIP)

DIP recommends that high-level modules should depend on abstractions rather than concrete implementations. Dependency Injection, as seen in frameworks like Spring (Java) and Angular (JavaScript), is a common technique that follows DIP, making systems more flexible and easier to modify.



Alt text: Dependency inversion principle concept

10. KISS (Keep It Simple, Stupid)

The KISS principle emphasises simplicity in design and implementation. By focusing on straightforward solutions, developers working with languages like Python or Ruby can write code that is easier to read, maintain, and debug without sacrificing performance.



Alt text: Writing code that is easy to read, maintain, and debug

11. Code Readability and Maintainability

Readable code, with consistent formatting and meaningful variable names, is essential for collaboration and long-term maintenance. Languages like Python, known for its clean syntax, inherently promote code readability, reducing technical debt and enhancing team productivity.

```
[1]: def square(num):
      """Return the square of the given number."""
      return num ** 2

# Example usage
number = 5
result = square(number)
print(f"The square of {number} is {result}.")
```

The square of 5 is 25.

Alt text: Python code with clean syntax

12. Performance Optimisation

Performance optimisation involves improving code efficiency and eliminating bottlenecks. In languages like C++ and Java, techniques such as algorithm optimisation, memory management, and parallelisation can significantly enhance performance and scalability.

Algorithm Efficiency

Effectiveness of algorithms in speed and resource use.

Data Structures

Choices affecting data organization and access speed.

Memory Management

Strategies for efficient memory use and allocation.

Parallelism and Concurrency

Simultaneous task execution for improved performance.

I/O Operations

Efficiency of input/output processes and data transfer.

Alt text: Factors affecting performance optimisation

13. Testing and Quality Assurance

Testing ensures software correctness and robustness. In languages like Python and JavaScript, automated testing frameworks such as PyTest or Mocha enable developers to validate different aspects of their software early in the development process, reducing defects.



Alt text: Software Quality Assurance concept

14. Version Control and Collaboration

Version control systems like Git are essential for tracking changes and collaborating on code. Branching and merging allow teams to work on different features simultaneously, ensuring code integrity and supporting efficient collaboration across development teams.



Alt text: Version control

15. Documentation and Knowledge Sharing

Good documentation clarifies system design and intent. By providing comprehensive documentation in languages like C# or Python, teams can streamline onboarding and ensure that future development and maintenance are aligned with the original design and functionality.



Alt text: Good documentation for system design