When to use Solid Principles

The SOLID principles are a set of design principles that guide software developers in creating well-structured, maintainable, and extensible code. These principles are intended to improve the quality of software by promoting modularity, flexibility, and ease of maintenance. Here's when you should use the SOLID principles:

1. **Single Responsibility Principle (SRP):**

Use SRP when you want to ensure that each class or module has a single, well-defined responsibility. This principle helps you avoid tightly coupled and difficult-to-maintain code by promoting clear separation of concerns.

2. **Open/Closed Principle (OCP):**

Use OCP when you anticipate changes or extensions in your software's functionality. By designing classes to be open for extension but closed for modification, you can introduce new behavior without modifying existing code.

3. **Liskov Substitution Principle (LSP):**

Use LSP when you're working with inheritance and subclasses. Adhering to LSP ensures that derived classes can be used interchangeably with their base classes without causing unexpected behavior or violating the contract.

4. **Interface Segregation Principle (ISP):**

Use ISP when you're designing interfaces to ensure that they remain cohesive and focused. By avoiding "fat" interfaces with too many methods, you can prevent clients from being forced to implement methods they don't need.

5. **Dependency Inversion Principle (DIP):**

Use DIP to achieve decoupling between high-level and low-level modules. Applying DIP can make your codebase more flexible and adaptable to changes, as high-level modules depend on abstractions rather than concrete implementations.

Overall, you should apply the SOLID principles whenever you want to create code that is:

- **Maintainable:** The principles help you design code that is easier to understand, modify, and extend over time.

- **Flexible:** By adhering to SOLID, your codebase becomes more adaptable to changes and new requirements.

- **Testable:** The principles promote modular and loosely coupled code, which makes it easier to write unit tests.

- **Reusable:** SOLID encourages the creation of small, focused components that can be reused in different contexts.

It's important to note that while the SOLID principles provide valuable guidelines for designing software, applying them rigorously in every situation might not always be necessary or practical. The key is to strike a balance between adhering to the principles and considering the specific needs and constraints of your project.