It seems like there might be a typo or a misunderstanding in your question. I'm not sure what you mean by "sold principal." However, I'll provide an explanation of the "SOLID" principles, which are a set of five design principles for writing maintainable and scalable software:

SOLID is an acronym that stands for:

1. **Single Responsibility Principle (SRP):**
   * A class should have only one reason to change, meaning it should have only one responsibility.
   * This principle encourages breaking down complex classes into smaller, focused classes that handle specific tasks.
2. **Open/Closed Principle (OCP):**
   * Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification.
   * New functionality should be added through extension rather than altering existing code.
3. **Liskov Substitution Principle (LSP):**
   * Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.
   * Subtypes must be substitutable for their base types without causing issues or violating the expected behavior.
4. **Interface Segregation Principle (ISP):**
   * A class should not be forced to implement interfaces it doesn't use.
   * This principle encourages creating smaller and more focused interfaces tailored to the needs of the implementing classes.
5. **Dependency Inversion Principle (DIP):**
   * High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions.
   * This principle promotes loose coupling by ensuring that higher-level modules depend on abstract interfaces or classes rather than concrete implementations.

The SOLID principles were introduced by Robert C. Martin as guidelines for writing maintainable, flexible, and easy-to-understand software. Following these principles can lead to code that is easier to refactor, extend, and maintain over time. Each principle addresses a specific aspect of software design, and together they contribute to creating well-structured and robust software systems.