### **Principes SOLID**

1. **Single Responsibility Principle (SRP)**
   * Un objet ou une classe doit avoir une seule raison de changer, c’est-à-dire une seule responsabilité ou tâche.
2. **Open/Closed Principle (OCP)**
   * Les entités logicielles (classes, modules, fonctions, etc.) doivent être ouvertes à l’extension mais fermées à la modification.
3. **Liskov Substitution Principle (LSP)**
   * Les objets d’une sous-classe doivent pouvoir remplacer les objets de la superclasse sans altérer le comportement souhaité du programme.
4. **Interface Segregation Principle (ISP)**
   * Les clients ne devraient pas être contraints de dépendre d’interfaces qu’ils n’utilisent pas ; une interface doit être spécifique à un client.
5. **Dependency Inversion Principle (DIP)**
   * Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau ; les deux devraient dépendre d’abstractions (comme des interfaces), et les abstractions ne devraient pas dépendre des détails, mais les détails devraient dépendre des abstractions.

### **Anti-patterns STUPID**

1. **Singleton**
   * Le pattern Singleton force la création d'une seule instance d'une classe, ce qui peut nuire à la testabilité et à la flexibilité.
2. **Tight Coupling**
   * Les classes fortement couplées sont dépendantes les unes des autres, ce qui rend le code difficile à modifier ou à tester.
3. **Untestability**
   * Code difficile à tester en raison de dépendances rigides ou d’une structure inadaptée.
4. **Premature Optimization**
   * Optimiser le code avant que cela ne soit nécessaire, ce qui peut compliquer la conception sans bénéfice immédiat.
5. **Indescriptive Naming**
   * Utiliser des noms peu descriptifs pour les variables, fonctions ou classes, ce qui rend le code difficile à comprendre.
6. **Duplication**
   * Copier-coller du code au lieu de le refactorer en méthodes ou classes réutilisables.