Single Responsibility

> *Each software module or a class should have one and only one reason to change.*

Benefits	Smells
• Separated classes can be reused in other parts of an application • Separated classes can be easily tested separately	• More than one contextually separated piece of code within single class • Large setup in tests

Open/Closed

> *Entities in your software system should be open for extension, but closed for modification.*

Benefits	Smells
• Functionality can be easily extended without affecting base • Code is loosely coupled and easily mocked	• Inheritance with different purposes • Complex if/else or switch statements

Liskov Substitution

> *Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.*

Benefits	Smells
• More intuitive and predictable behaviour • Clear distinction between shared interface and extended functionality	• Modification of inherited behaviour in subclass • Exceptions raised in overridden inherited methods

Interface Segregation

> *Clients should not have to implement interfaces that they don't use.*

Benefits	Smells
• Further decoupling of all implementing classes • Clear separation of business logic	• Interface consists of many properties or methods • Methods and properties are not always implemented in classes

Dependency Inversion

> *High level modules should not depend on lower level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.*

Benefits	Smells
• High level modules are independent of low-level modules, increasing reusability • Injected classes are easily mocked in tests	• Instantiation of low-level classes • Calls to class methods rather than interface methods