expoQA 24

MADRID
May 28th, 29th, 30th 2024

sogeti
expo:QA
International Conference on Software Testing & Quality Engineering

expoqa.com

Building for excellence:

Test Design Patterns for sustainable automation framework

WELCOME!

Toni Robres

Domain Tech Lead SCRM

Passionate about Quality and software Development

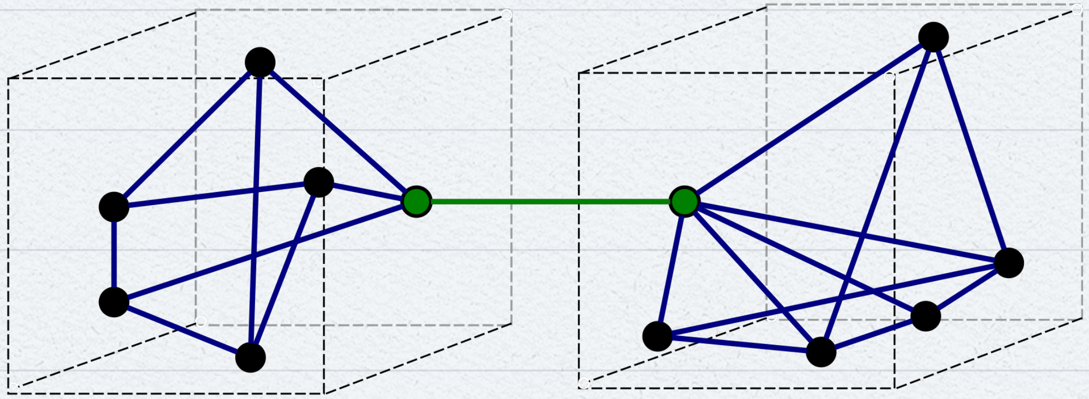
亀

What you will learn today

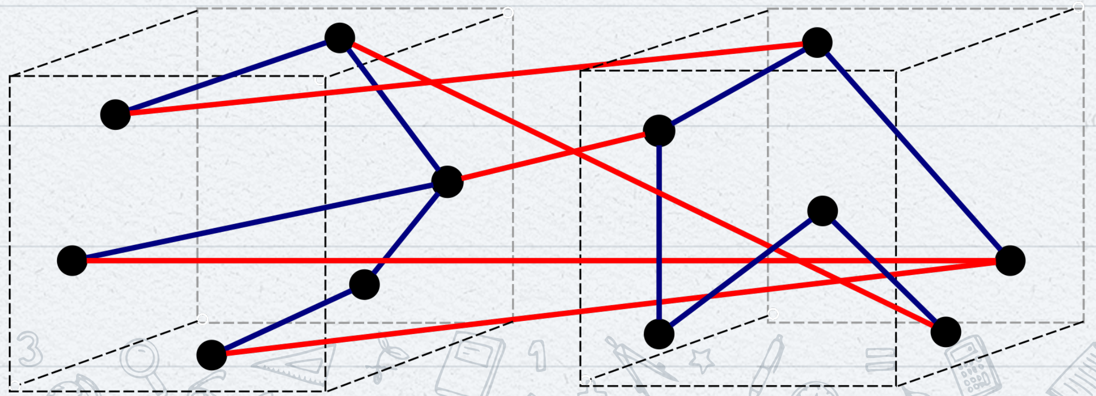
- ✗ Introduction
- ✗ SOLID principles
- ✗ Design Patters

Coupling and cohesion

- Resolving complex problems → Modularization
- The way we structure the modules and how they are interacting to each other give rise to:
  - Coupling
  - Cohesion



a) Good (loose coupling, high cohesion)



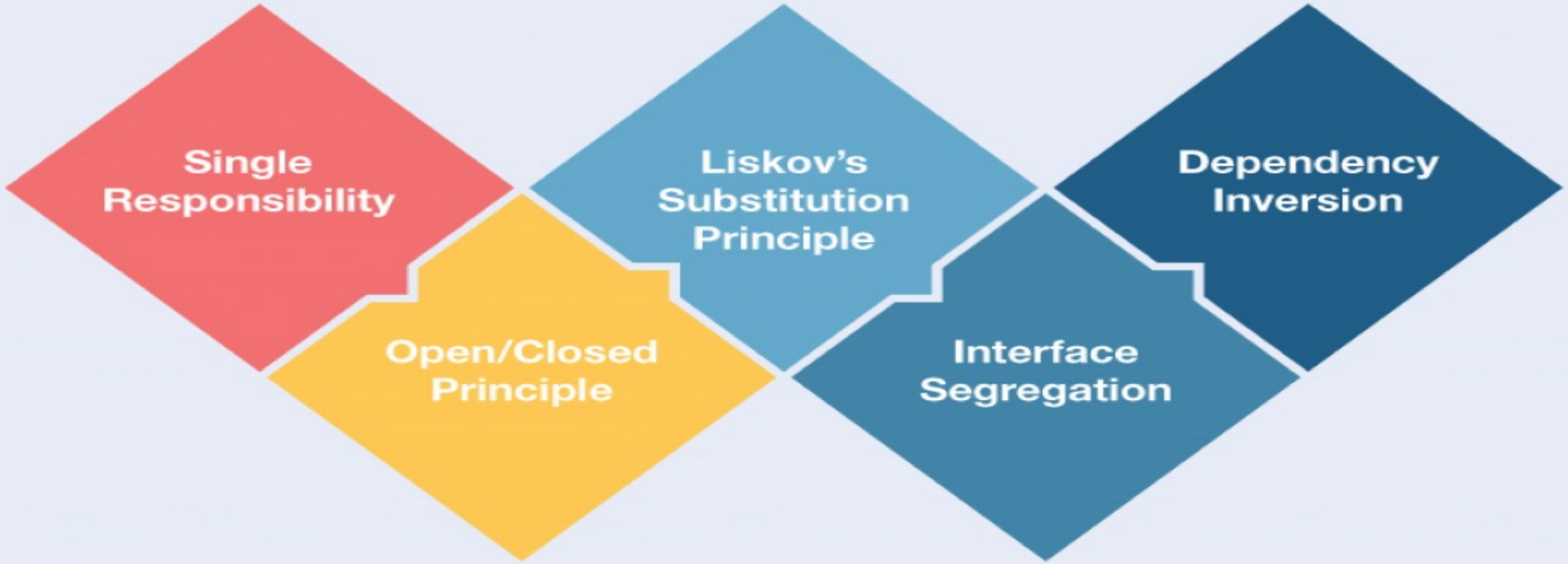
b) Bad (high coupling, low cohesion)

Coupling and cohesion

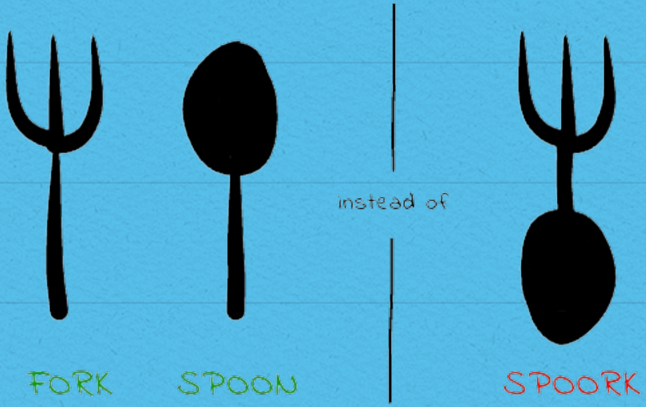
Low coupling and high cohesion advantages

- ✗ Improve maintainability
- ✗ Improve code reusing
- ✗ Improve readability
- ✗ Facilitate unit and integration tests

S.O.L.I.D.



Single Responsibility
Open/Closed Principle
Liskov's Substitution Principle
Interface Segregation
Dependency Inversion



Single Responsability

Any module should only be responsable for one (and only one) function

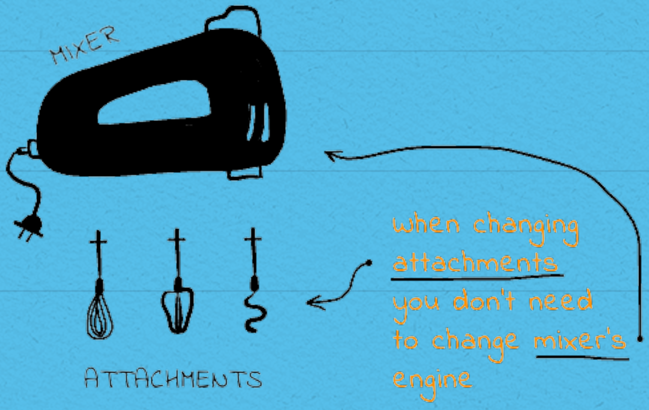
Single responsability

Idea

- Any class should have only one reason to change
- All services should be closely aligned with his responsability

Benefits

- High cohesion
- Allows class composition
- Avoid duplicities


MIXER
ATTACHMENTS
when changing attachments you don't need to change mixer's engine

Open / Close

Any entity should be open to extend and close to modify

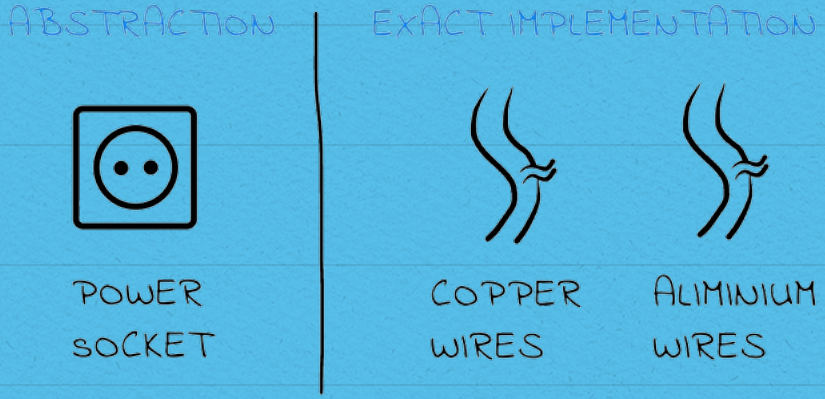
Open / Close

Idea

- A new feature not should cause a modification
- Can be aplicable to classes, services or use cases
- Avoid relying on specific implementations

Benefits

- Facilitate adaptation to new changes
- Avoid errors when the behaviour is modified



Dependency inversion

High level modules should not depend on low level modules

Dependency inversion

Idea

- High level modules should not depend on low level modules. Both should depend on abstractions
- The abstractions should not depend on detail specifications. The details should depend on abstractions

Benefits

- Facilitate implementation modifications
- It’s easy to substitute implementations
- Improve the class testability
- Visualize all dependencies

Software Engineering
Iterator
Command
Mediator
Factory
Refactoring
Repository
Proxy
Dependency
Injection
Adapter
Behavioral
Classes
Decorator
Chain
Code Analysis
Design
Observer
Singleton
Structural
Creational
Bridge
Composite
Patterns
Façade
Abstract Factory
Strategy
Agile Method
Test Driven Development
Template Method

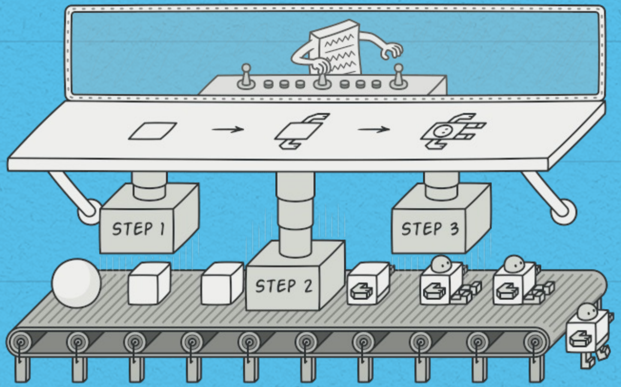
Design Patterns

What are?

- ✗ General reusable solution that can be applicable to different problems
- ✗ Templates to deal a general problems
- ✗ Focus on specific purposes
- ✗ Avoid reinventing the wheel

What are not?

- ❖ Are not copy paste
- ❖ Not aplicable to all the problems
- ❖ Not a silver bullet



Builder pattern

Building complex objects step by step

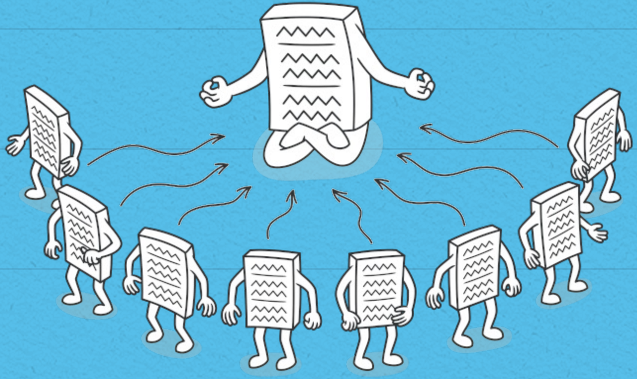
Builder pattern

Idea

- Make it simple the complex object building in simple steps
- Create different representations of complex objects

Advantages

- Provide control over the steps of the object building process
- Allow to vary an object internal representation.
- Increase readability in the builiding process



Singleton pattern

Only single instance of a class

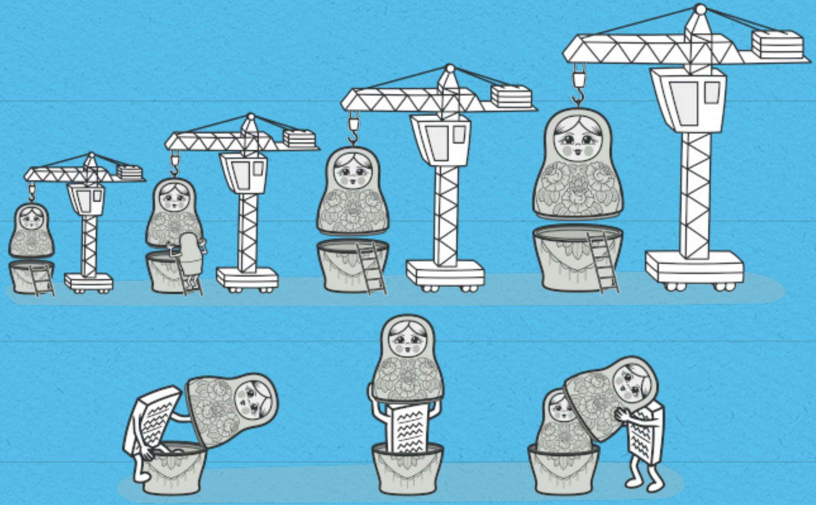
Singleton pattern

Idea

- ❖ Ensure that only one instance exists
- ❖ Provide unique Access point to the instance
- ❖ WARNING: be carefull multithreading

Advantatges

- ✗ Unique access to a specific class
- ✗ Avoid global variables
- ✗ Global access
- ✗ Lazy Initialization



Decorator pattern

Extends the object behaviour

Decorator pattern

Idea

- Add new behaviours to an individual object
- The behaviour is extended with wrappers with new functionalities

Advantatges

- ✗ Extend the class behaviour without create new objects
- ✗ Modify the responsabilities during runtime
- ✗ Is possible combine several decorators using different decorators

Factory pattern

Create objects without having to specify their exact class

Factory pattern

Idea

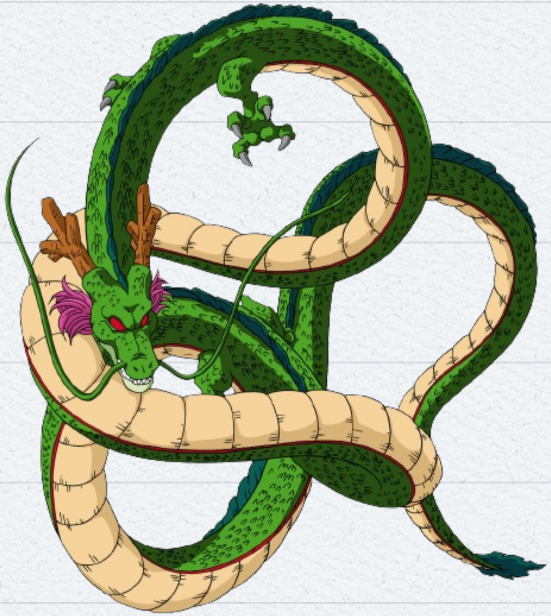
- A factory class Will be the responsable to créate different objects

Advantatges

- ✗ Allow create objects without tightly coupling to specific classes
- ✗ Allow extends new module types without modify the remaining modules

Summary

- ✗ Reduce coupling and increase cohesion
- ✗ SOLID as values
- ✗ Desing patterns to improve the test architecture



THANKS!

Twitter: @twiindan

Email: twiindan@gmail.com

Linkedin: www.linkedin.com/in/antoniorobres/

Github: https://github.com/twiindan/SOLID_patterns expoQA® 24

MADRID
May 28th, 29th, 30th 2024

Thank you for attending

expoqa.com