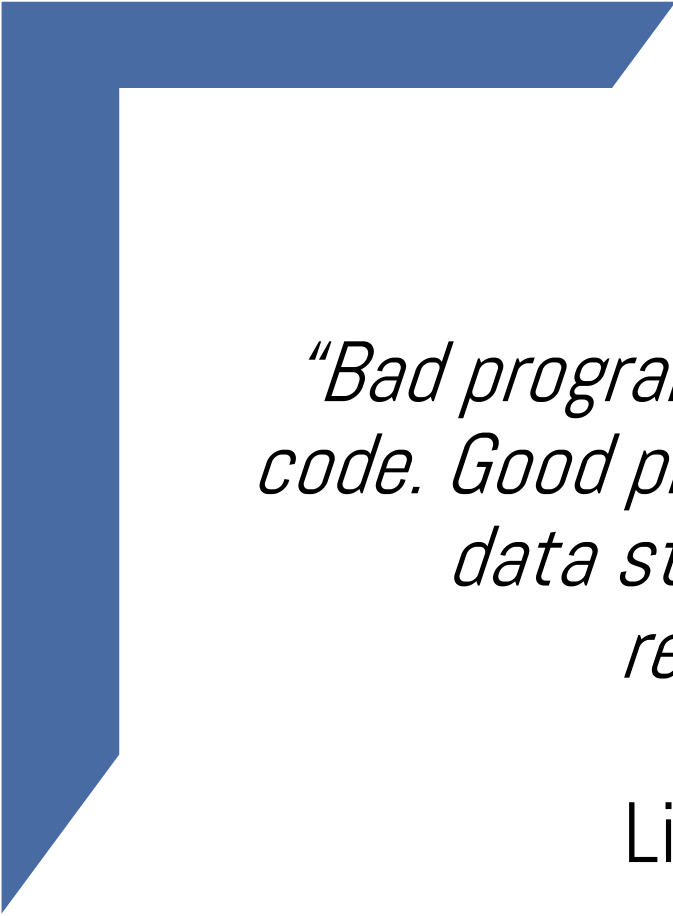# Solid and clean

Paolo Burgio
paolo.burgio@unimore.it

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

Linus Torvald

What will we see?

› ~~How to implement patterns within a given language~~

› ~~How to choose variables' names~~

› ~~CamelCase vs snake_case~~

› ~~Best coding practices~~

How to code properly

› "Properly?"

› In such a way that your code becomes scalable, maintainable, robust..

› Do engineer's job!

SOLID programming

Aka: **Object Oriented Design**

Five principles that ~~save your life~~ make the difference between a programmer/coder and a software architect

> (Or between an happy person and a sad person)

Applicable to object-oriented programming (but not only)

1. **S**ingle Responsibility
2. **O**pen/Close principle
3. **L**iskov substitution
4. **I**nterface segregation
5. **D**ependency inversion

Single responsibility principle

A software entity should have only one reason to change

› Aka: every class should have a single responsibility or single job or single purpose

› **Answers to**: "What should I put into a class?"

› **Pros**: you always know where/what/how to change your code, and don't mess up things

› (**Cons**: increase number of classes and effort…)

Open/Close principle

Entities should be open for extension, but closed for modification

› Aka: you should never change anything, always adding new behavior using polymorphism

› **Answers to**: "How should I extend my code?"

› **Pros**: reduce the number of bugs and headaches

› (**Cons**: N/A)

Liskov substitution principle

You should be able to substitute any parent class with any of their children without any behavior modification

› Aka: <u>I lied you!</u>
  – (Rectangles vs Squares)

› **Answers to**: "When and what should I inherit?"

› **Pros**: code is scalable, and minimize changes upon modifications

› (**Cons**: you have to think before you code)
  – not actually a con…

Interface segregation principle

Many client specific interfaces are better than a big one

› Aka: Interfaces should be the minimal set of behaviors you need

› **Answers to**: "When should I create an interface?"
  – Spolier: "as much as you can"

› **Pros**: you minimize dependencies in your code (thus, programming effort)

› (**Cons**: trust me…NONE)

Always remember: an interface is a contract. You can build (automatic) tests over contracts!

Dependency inversion principle

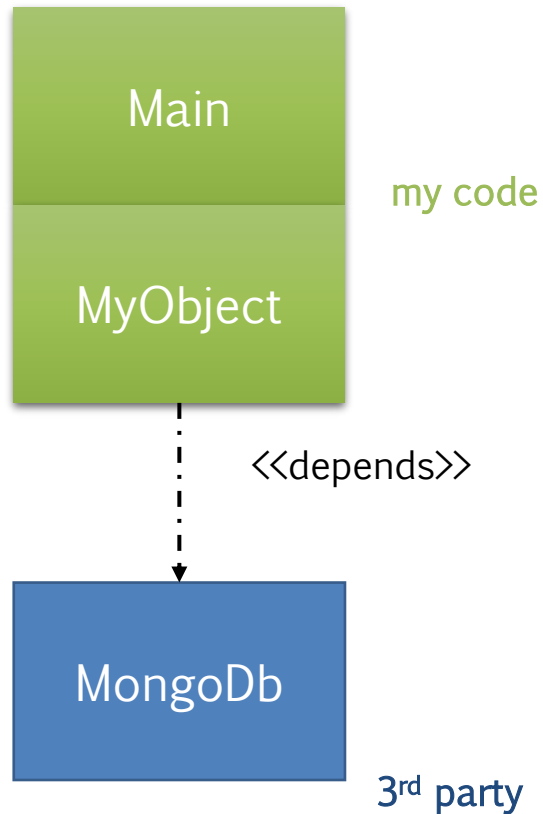*Your project shouldn't depend of anything, make those things depend of interfaces*

› Design wrappers around your dependencies
  – (This is **NOT** "dependency injection"…but its good friend)

› **Answers to**: "How can I avoid getting crazy with dependencies?"

› **Pros**: isolation between code components; your code reflects the analysis/model of business

› (**Cons**: additional programming effort)
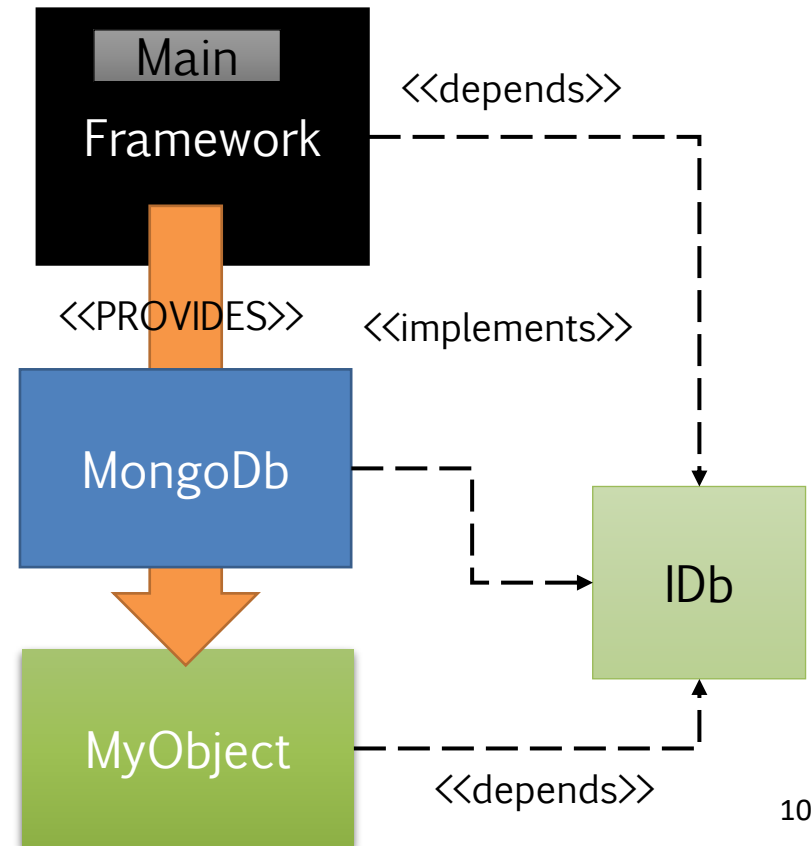
Dependency inversion principle

Library

› Tied to 3rd party code

```
┌─────────────────┐
│      Main       │
│                 │   my code
│    MyObject     │
└────────┬────────┘
         ┊  <<depends>>
         ▼
┌─────────────────┐
│                 │
│    MongoDb      │
│                 │   3rd party
└─────────────────┘
```

Framework

› Inversion of control
› Dependency injection

```
┌──────────────────────┐
│  ┌──────────────┐    │        <<depends>>
│  │    Main      │    │─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐
│  └──────────────┘    │                    │
│     Framework        │                    │
└──────────┬───────────┘                    │
       <<PROVIDES>>   <<implements>>         │
           │                                 │
           ▼                                 │
┌──────────────────────┐                     │
│                      │ ─ ─ ─ ─ ─ ┐         │
│      MongoDb         │           │    ┌────┴──────┐
│                      │           └──► │           │
└──────────┬───────────┘                │    IDb    │
           │                            │           │
           ▼                            └────▲──────┘
┌──────────────────────┐                     │
│                      │ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
│      MyObject        │       <<depends>>
│                      │
└──────────────────────┘
```
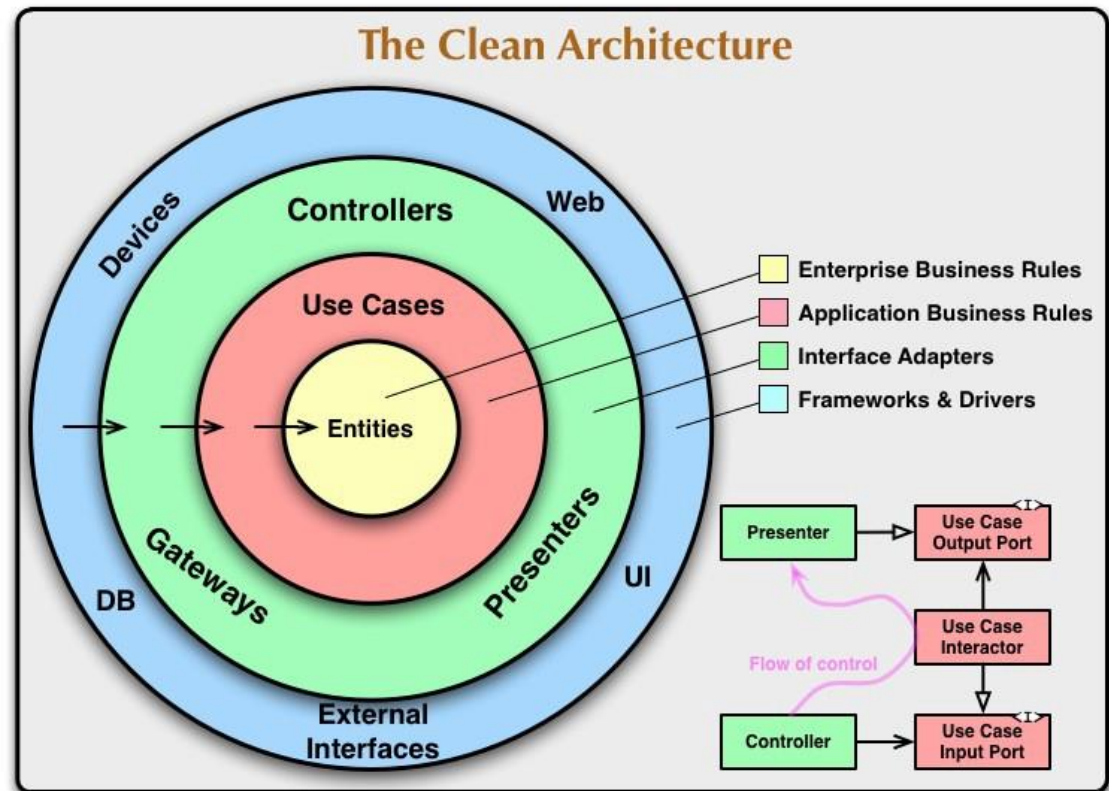
10

CLEAN architecture

› Applies to big projects

› Specifies how to model your problem(s)..and how to manage implementation

› Enables scalable, maintainable, and easy-to-test codebase

› (OF course) builds upon SOLID programming style

What does it mean?

The Clean Architecture

Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

› Dependencies go inside, never outside!

› Enabled by a thorough analysis/modeling of the problem

› You shall not become a slave of "How you do things"
  – (webpages, Ethernet..)

References

Course website

› http://hipert.unimore.it/people/paolob/pub/Industrial_Informatics/index.html

My contacts

› paolo.burgio@unimore.it

› http://hipert.mat.unimore.it/people/paolob/

Resources

› https://springframework.guru/solid-principles-object-oriented-programming/

› https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/

› A "small blog"
  – http://www.google.com