“You can use an eraser on the drafting table or a sledgehammer on the construction site.”

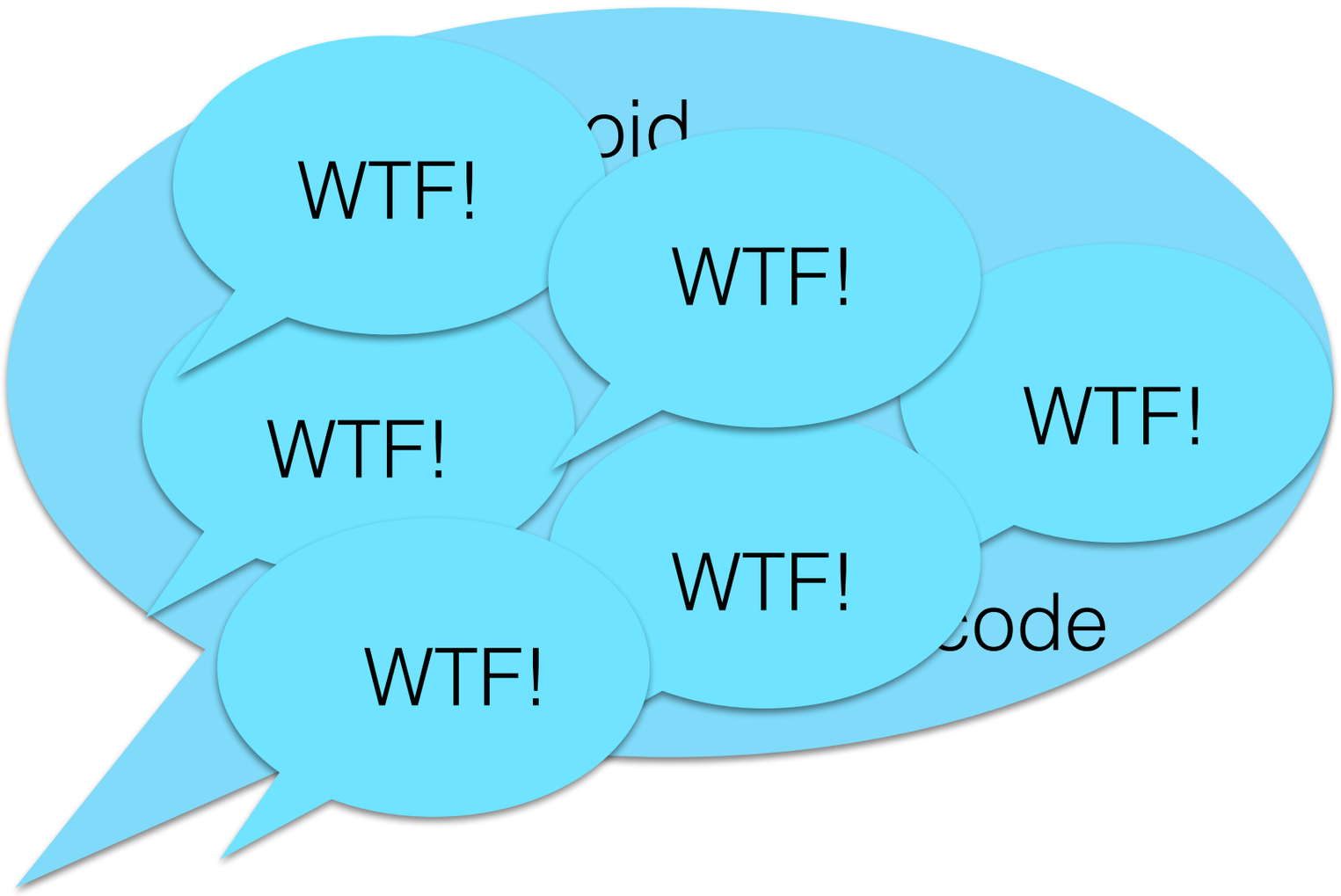
–Frank Lloyd Wright

What is object-oriented design?

- Requirements translated into modules, classes, methods/functions and interactions between them.

- The design process often results in a program structure that can handle change. Remember, “change is the only constant”.

How do we know a design is bad?



Ok, we probably need better criteria :)

Rigidity

- System is hard to change.

- Every new change has a cascading effect on other parts of the system.

Fragility

- When a change is made, it renders the system to break in unrelated areas.

- Tests fail in unchanged parts of code. Fixing those problems leads to more problems.

Immobility

- Hard to reuse components of the system that could be useful elsewhere too.

- Effort and risks involved in extracting them result in inability to reuse.

Needless Repetition

- Copy-pasted code “inspired” from other sources.

- Calls for a need of an abstraction to take care of the repetitive tasks.

Viscosity

- Multiple ways to make a change. Doing it in a hackish way is easier than preserving the design.
- Doing things right is harder than doing things wrong.

Opacity

- Code works, no idea why. Code doesn’t work, no idea why.

- Code was easier to write; not easier to read.

Needless Complexity

- Designing more than required without any extra benefit.

- Unused code. Maintenance nightmare.

How do we achieve a good design?

SOLID!!

SOLID

- An acronym of acronyms!
- It recalls in a single word all the most important principles of design
  - SRP: Single Responsibility Principle
  - OCP: Open Closed Principle
  - LSP: Liskov Substitution Principle
  - ISP: Interface Segregation Principle
  - DIP: Dependency Inversion Principle

SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

SRP

- A class should have only one reason to change.

- Additional responsibilities should be moved to their own classes.

OPEN CLOSED PRINCIPLE

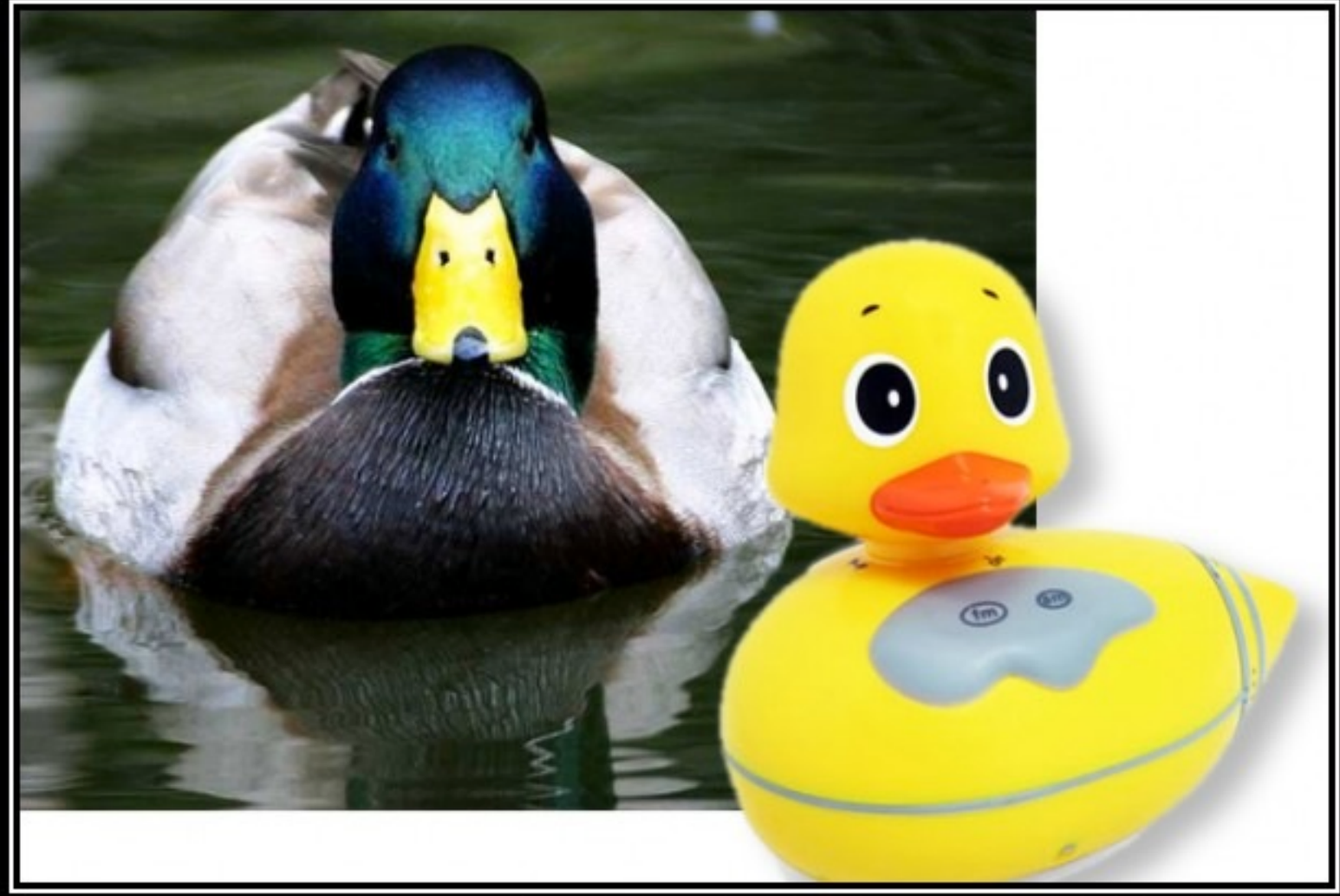
Open Chest Surgery Is Not Needed When Putting On A Coat

OCP

- Open for Extension: Behavior of a module can be extended. New behaviors can be added.

  Closed for Modification: The source code of such a module should be untouched.

- Polymorphism is the key.



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

LSP

- Derived types should be substitutable for their base types without breaking client's expectations of the base type.

- Think twice before subtyping. If there are related behaviors, think about extracting to a common supertype instead of deriving one from other.

INTERFACE SEGREGATION PRINCIPLE

There is no combined power & water connector

ISP

- Instead of one fat interface use multiple skinny cohesive interfaces.

- Clients should not depend on interface methods that they don’t need to use.

DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

DIP

- High level modules should not depend on low level modules. Both should depend on abstractions.

- Clients should own the interfaces. The server implementation should depend on the client needs.