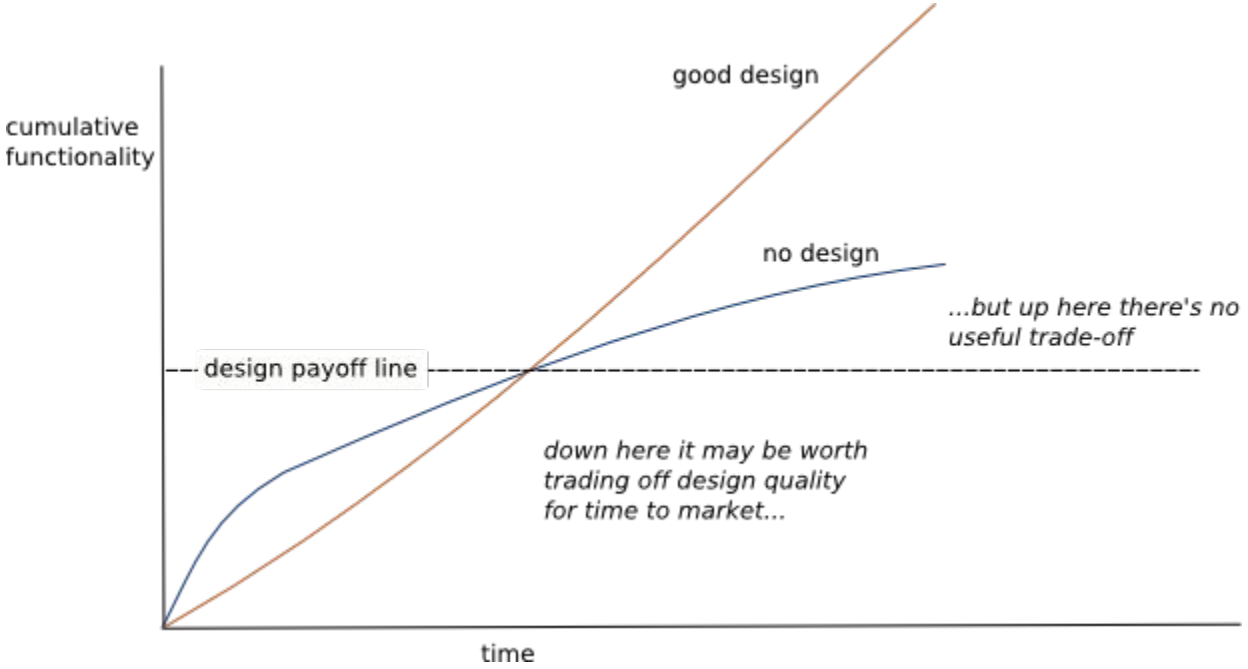
Принципы проектирования и дизайна ПО

Лекция №6

Агошков Илья 2016

Design stamina hypothesis

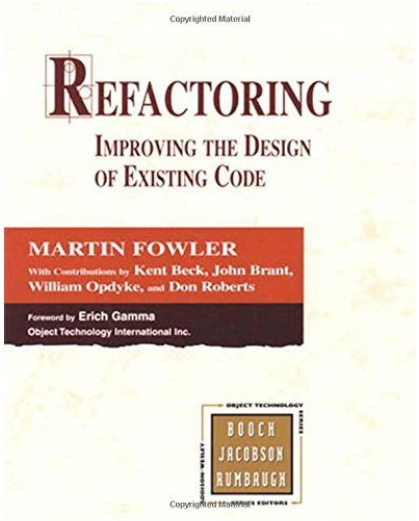


SOLID

Software Development is not a Jenga game

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

Martin Fowler, "Refactoring: Improving the Design of Existing Code"



Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on **explaining to human beings what we want a computer to do**.

The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. He or she strives for a **program that is comprehensible** because its concepts have been introduced in an order that is **best for human understanding**, using a mixture of formal and informal methods that reinforce each other.

Donald Knuth

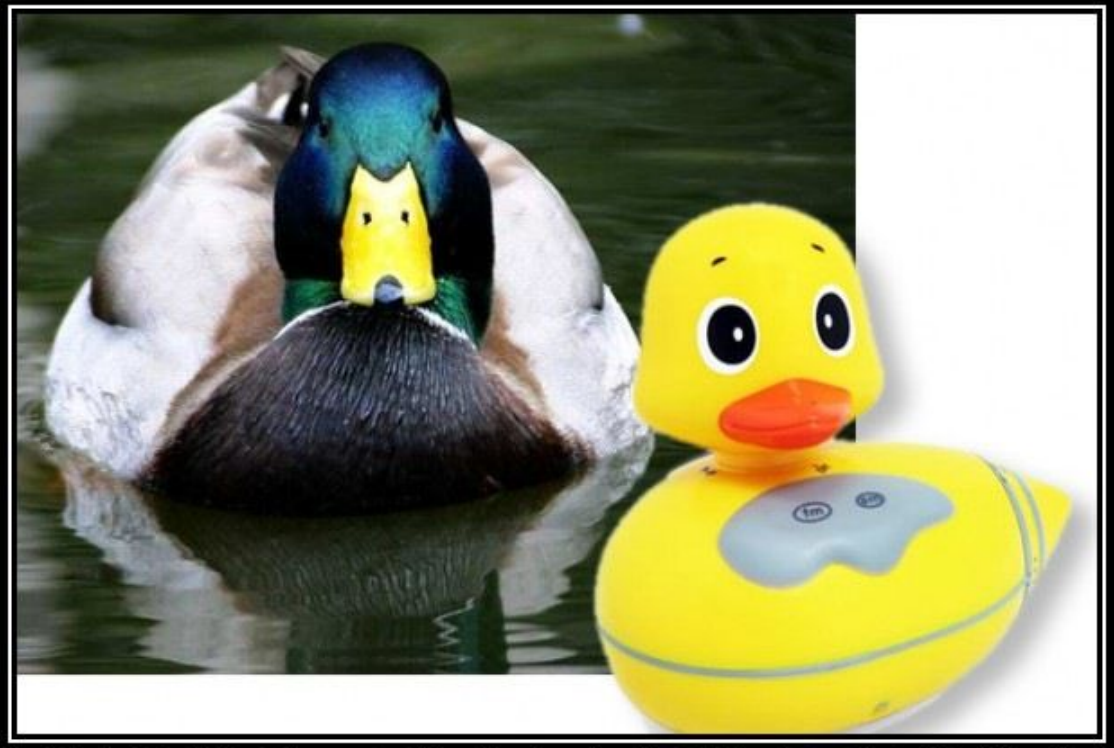
SOLID principles

Single responsibility principle
Open-closed principle
Liskov substitution principle
Interface segregation principle
Dependency inversion principle

Liskov substitution principle

What is wanted here is something like the following substitution property: If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T

Barbara Liskov



LISKOV SUBSTITUTION PRINCIPLE

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

Substitutability

Child classes must not:

- remove base class behavior
- violate base class behavior

LSP and inheritance

Inheritance = IS-A relationship

LSP suggests IS-A should be replaced with IS-SUBSTITUTABLE-FOR

LSP violation “smells”

```
for (Employee employee : employees) {
  if (employee instanceof Manager) {
    printer.printManager((Manager) employee));
} else {
    printer.printEmployee(employee);
}
```

LSP violation “smells”

```
public abstract class Base
{
  public abstract void method1();
  public abstract void method2();
}
```

LSP violation “smells”

```
public Child extends Base
{
  public void method1() {
    throw new NotImplementedException();
  }
  public void method2() {
    // do something useful
  }
}
```

LSP tips

“Tell, don’t ask”

- don’t look inside objects for their internals -- move behavior to the object
- tell the object want you want it to do

Consider refactoring to a new base class

- given two classes that share a lot but are not substitutable
- extract a new base class
- derive both classes from a new base and ensure substitutability is there

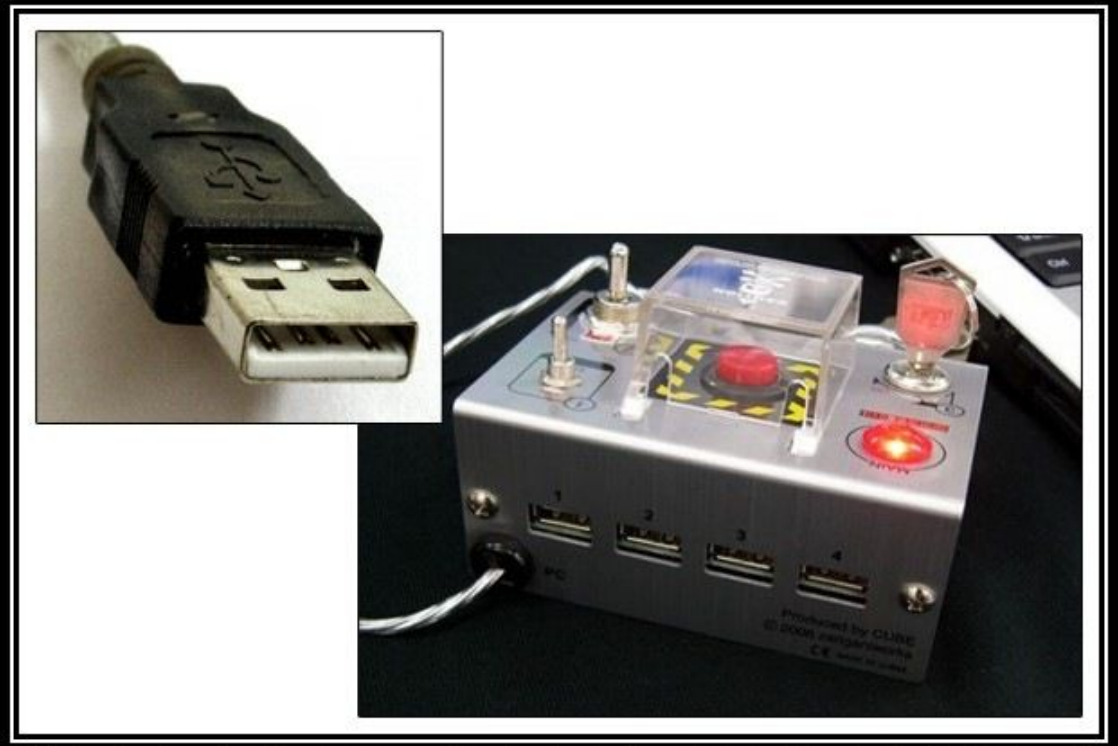
LSP summary

Conformance to LSP allows for proper use of polymorphism and produce more maintainable code

Remember IS-SUBSTITUTABLE-FOR instead of IS-A

Interface segregation principle

CLIENTS SHOULD NOT BE FORCED TO DEPEND UPON INTERFACES THAT THEY DO NOT USE



INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

ISP how to fix

If you find yourself depending on a “fat” interface you own

create smaller interfaces
have fat interface extend smaller interface
reference new interface from code

If you find yourself depending on a “fat” interface that you don’t own

create smaller interface with just what you need
implement this interface using an Adapter pattern