Bruno Bossola

SOLID Design Principles

GeeCON

Let's move the Java world!

About me

- C Developer since 1988
- Java developer since 1996
- XP Coach during 2000-2001
- Lead coordinator and co-founder of JUG Torino in 2001
- Sun Java Champion since 2005
- Speaker at Javapolis and Jazoon conferences

GeeCON
Let's move the Java world!

Agenda

- What design exactly is about?
- Bad design
- Good design
- SOLID principles
- Conclusions

GeeCON
Let's move the Java world!

Warning!

- Most of those slides maps directly various articles available to the web, expecially from objectmentor.com (thanks Bob!)
- Note that those articles are dated...

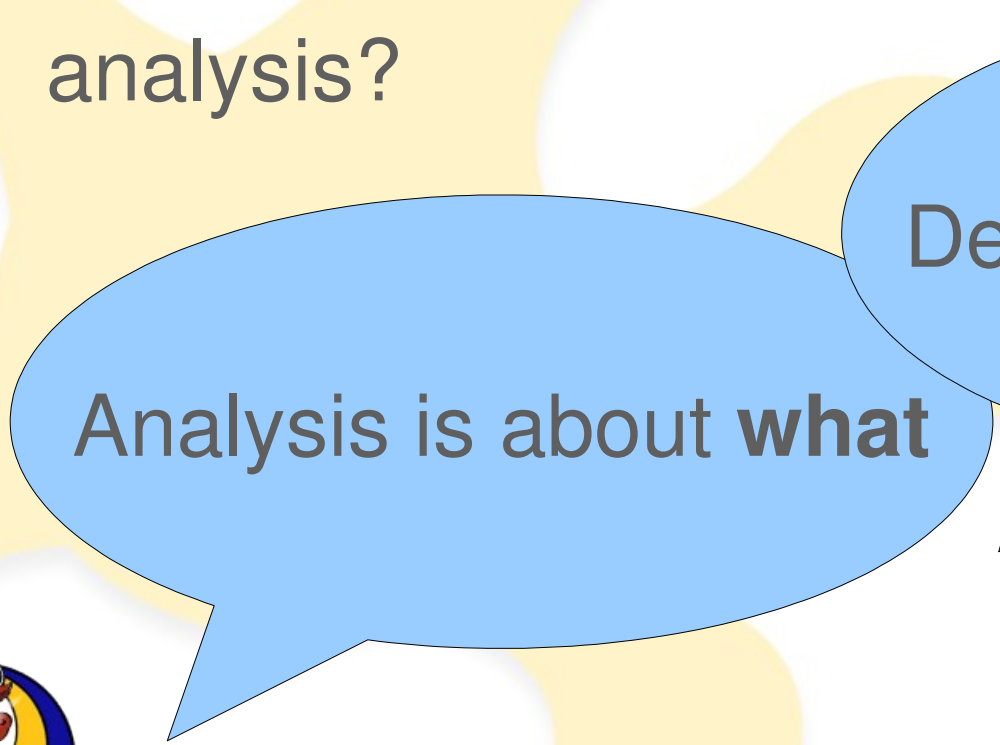
  1992-1998!!!

...so please don't force me to get here next year!!!!

GeeCON
Let's move the Java world!

Design

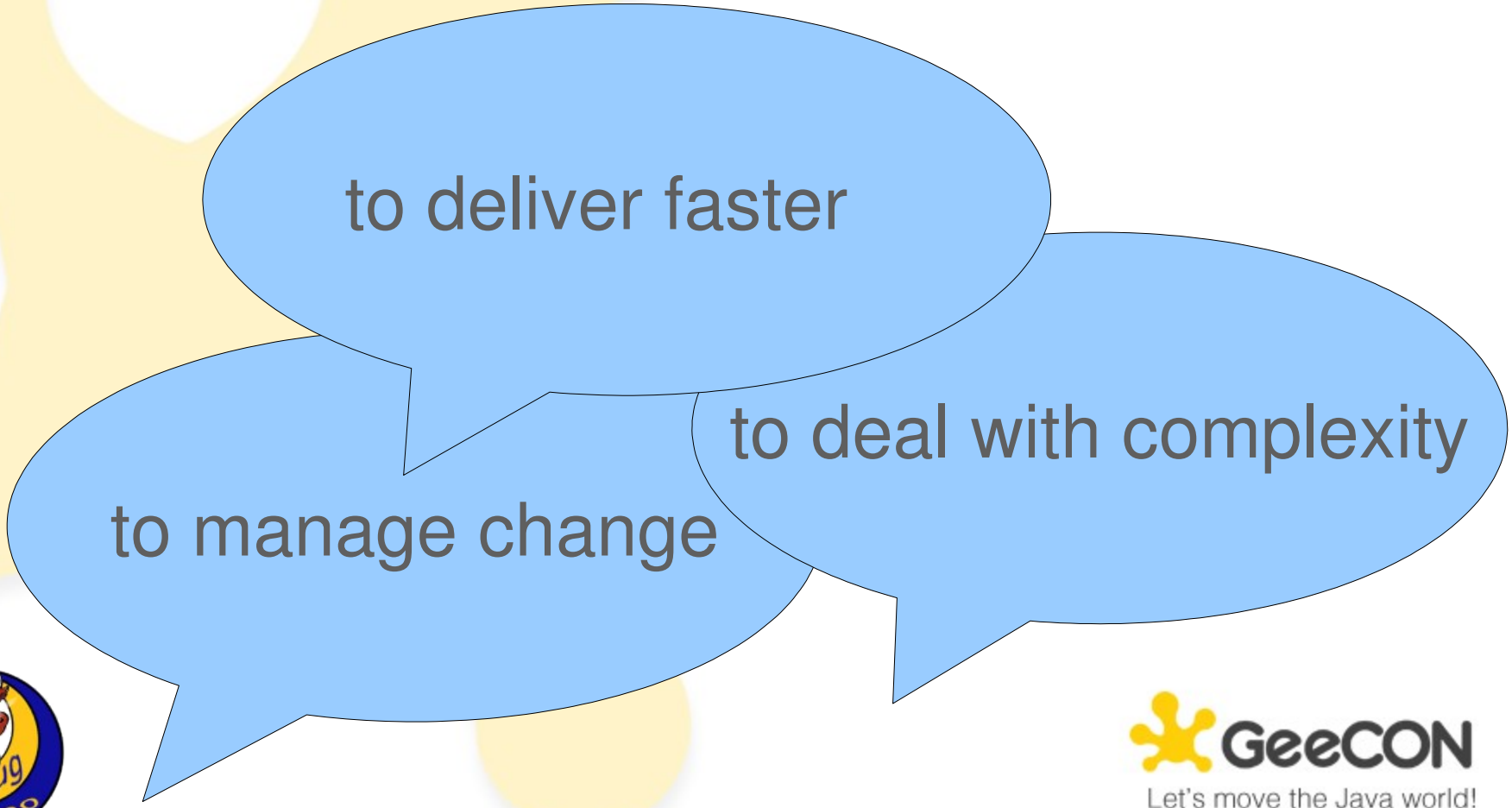
- What's the meaning of design?
- What's the difference if compared to analysis?



GeeCON
Let's move the Java world!

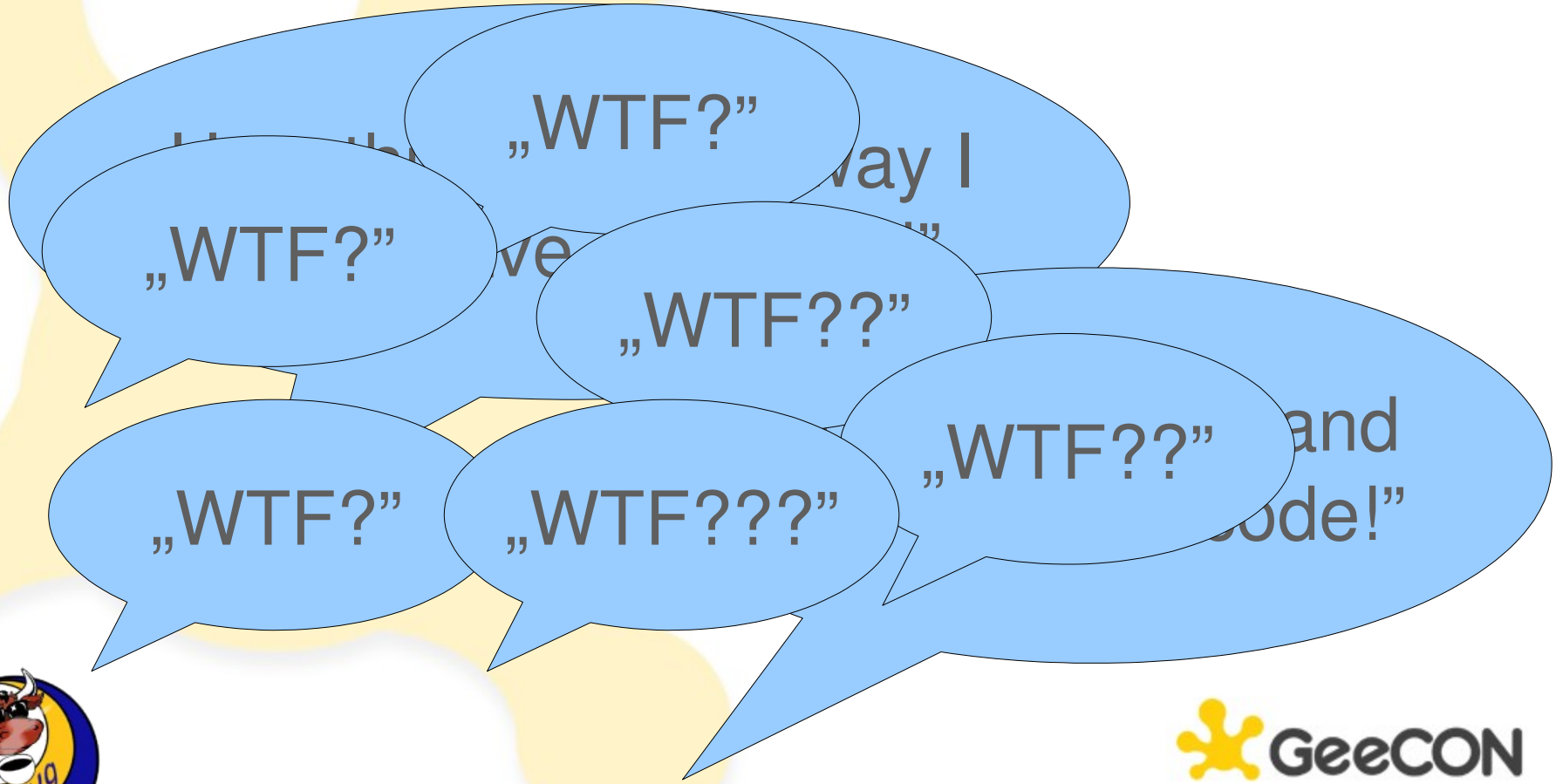
Design

- Why do we need (good) design?



Design

- How do we know a design is bad?



Design

- Ok, we probably need better criterias :)
- Are there any „symptoms” of bad design?

GeeCON
Let's move the Java world!

Rigidity

- **the impact of a change is unpredictable**
- every change causes a cascade of changes in dependent modules
- a nice „two days” work become a kind of endless marathon
- costs become unpredictable

GeeCON
Let's move the Java world!

Fragility

- **the software tends to break in many places on every change**
- the breakage occurs in areas with no conceptual relationship
- on every fix the software breaks in unexpected ways

GeeCON
Let's move the Java world!

Immobility

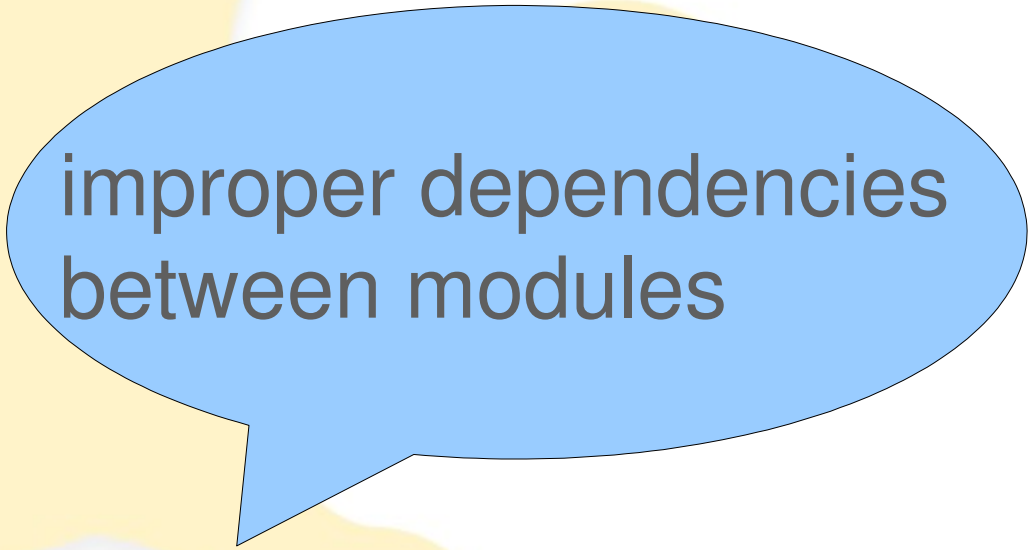
- **it's almost impossible to reuse interesting parts of the software**
- the useful modules have too many dependencies
- the cost of rewriting is less compared to the risk faced to separate those parts

Viscosity

- **a hack is cheaper to implement than the solution within the design**
- preserving-design moves are difficult to think and to implement
- it's much easier to do the wrong thing rather than the right one

Design

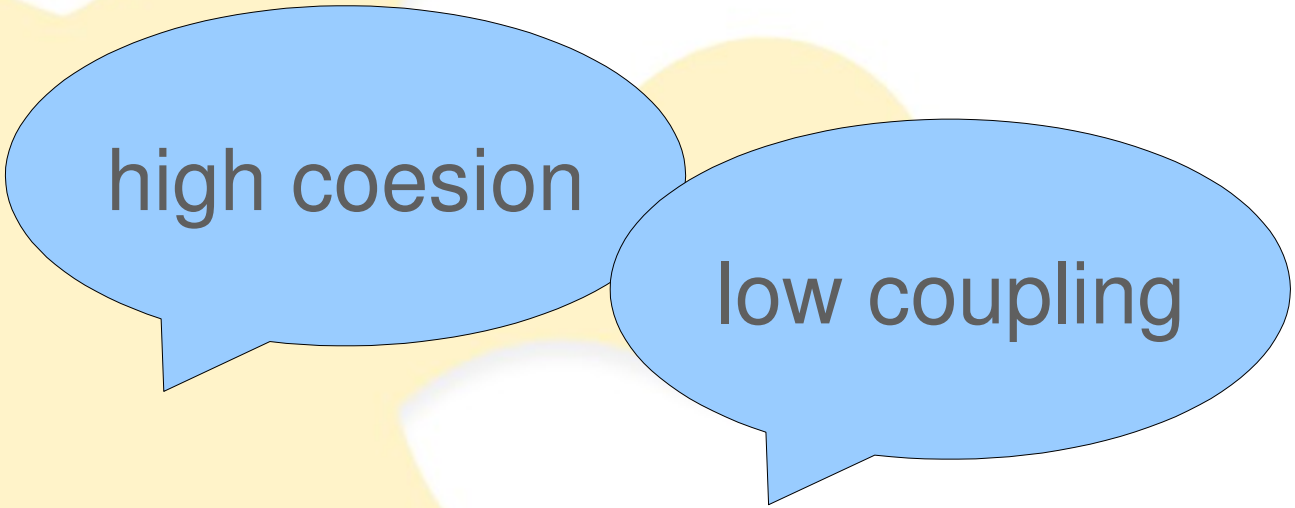
- What's the reason why a design becomes rigid, fragile, immobile, and viscous?



GeeCON
Let's move the Java world!

Good design

- So, what are the characteristics of a good design?



Good design

- How can we achieve a good design?

Let's go **SOLID**!!!

SOLID

Software Development is not a Jenga game

GeeCON

Let's move the Java world!

SOLID

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

BREAK!

- What is the best comment in source code you have ever encountered? (part 1)

```
/*
 * You may think you know what the following code does.
 * But you dont. Trust me.
 * Fiddle with it, and youll spend many a sleepless
 * night cursing the moment you thought youd be clever
 * enough to "optimize" the code below.
 * Now close this file and go play with something else.
 */
```

(source: stackoverflow.com)

GeeCON
Let's move the Java world!

SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

Single Responsability Principle

- **A software module should have one and only one responsability**
- Easier: a software module should have one reason only to change
- It translates directly in high coesion
- It's usually hard to see different responsabilities

GeeCON
Let's move the Java world!

SRP

- Is SRP violated here?

```
interface Modem
{
    public void dial(String pno);
    public void hangup();
    public void send(char c);
    public char recv();
}
```

GeeCON
Let's move the Java world!

SRP

- Is SRP violated here?

```
interface Employee
{
    public Pay calculate();
    public void report(Writer w);
    public void save();
    public void reload();
}
```

GeeCON
Let's move the Java world!

SRP

- identify things that are changing for different reasons
- group together things that change for the same reason
- note the bias compared to "classical" OO
- smells? *Manager, *Controller, *Handler
  - you really don't know how to name those
  - SRP requires very precise names, very focused classes

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

GeeCON

Let's move the Java world!

Open Closed Principle

- **Modules should be open for extension but closed to modification**
- theorized in 1998 by Bertrand Meyer in a classical OO book
- you should be able to extend the behavior of a module without changing it!

GeeCON
Let's move the Java world!

OCP?

```
void DrawAllShapes(ShapePointer list[], int n)
{
  for (int i=0; i<n; i++)
  {
    struct Shape* s = list[i];
    switch (s->type)
    {
      case square: DrawSquare((struct Square*)s);
                   break;
      case circle: DrawCircle((struct Circle*)s);
                   break;
    }
}
```

GeeCON
Let's move the Java world!

OCP?

```
public void draw(Shape[] shapes) {
    for( Shape shape : shapes ) {
        switch (shape.getType()) {
            case Shape.SQUARE:
                draw((Square)shape);
                break;
            case Shape.CIRCLE:
                draw((Circle)shape);
                break;
            }
        }
    }
```

OCP!

```
public void draw(Shape[] shapes) {
    for( Shape shape : shapes ) {
        shape.draw();
    }
}
```

GeeCON
Let's move the Java world!

OCP

- Abstraction is the key!

Uncle Bob's recipe

- keep the things that change frequently away from things that don't change
- if they depend on each other, things that change frequently should depend upon things don't change

GeeCON
Let's move the Java world!

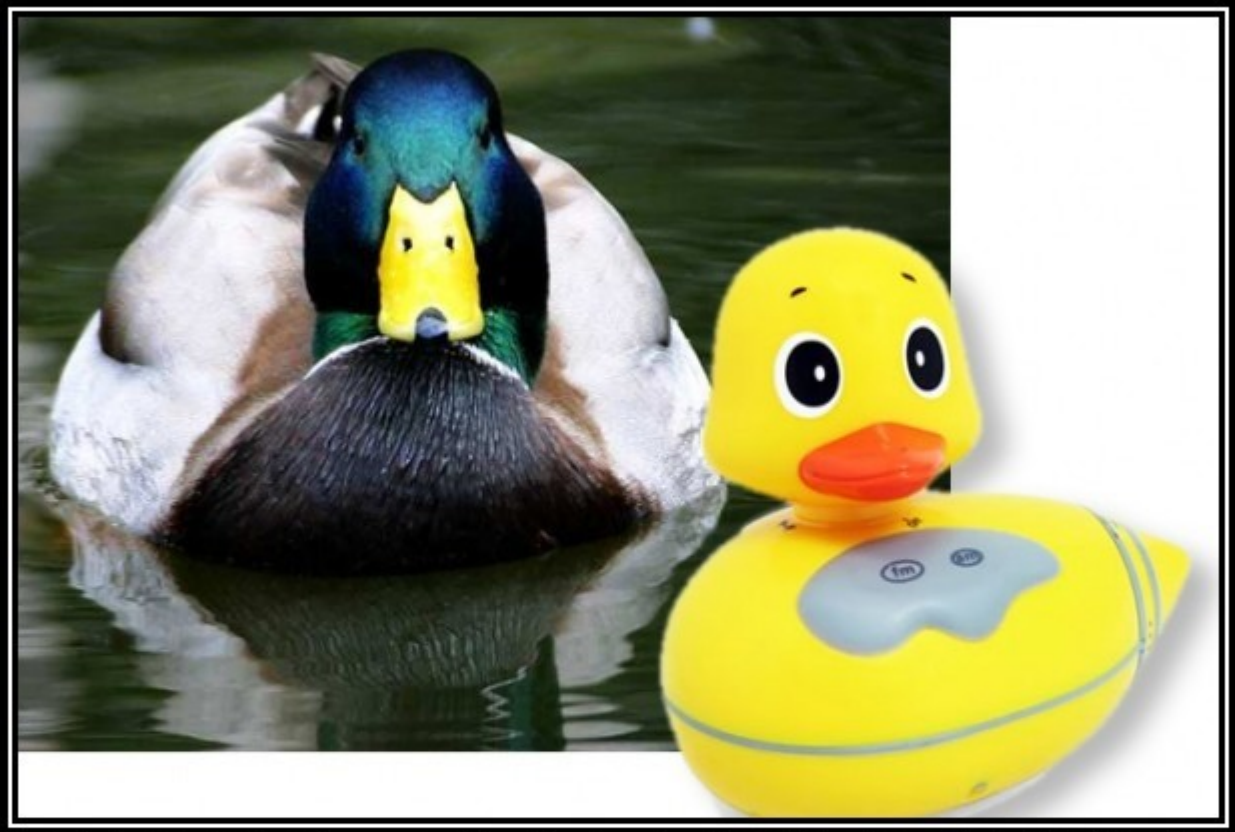
BREAK!

- What is the best comment in source code you have ever encountered? (part 2)

```
// I dedicate all this code, all my work, to my wife,
// Darlene, who will have to support me and our three
// children and the dog once it gets released into
// the public.
```

(source: stackoverflow.com)

GeeCON
Let's move the Java world!



LISKOV SUBSTITUTION PRINCIPLE

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

JUG Torino

GeeCON
Let's move the Java world!

Liskov Substitution Principle

- **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.**

- uh?

GeeCON
Let's move the Java world!

LSP

- Try #2:

- **Function that use pointers or references to base classes must be able to use objects of derived classes without knowing it**

GeeCON
Let's move the Java world!

LSP

- Try #3:

- **Given an entity with a behavior and some possible sub-entities that could implement the original behavior, the caller should not be surprised by anything if one of the sub entities are substituted to the original entity**

GeeCON
Let's move the Java world!

LSP (by example)

- How would you model the relationship between a square and a rectangle?
- Should the square class extends rectangle?

GeeCON
Let's move the Java world!

LSP (by example)

- Of course, isn't the Square a kind of Rectangle, after all?
- It seems an obvious ISA

GeeCON
Let's move the Java world!

LSP (by example)

- But... what about:
  - rectangle has two attributes, width and height: how can we deal with that?
  - how do we deal with setWidth() and setHeight() ?
- Is it safe?

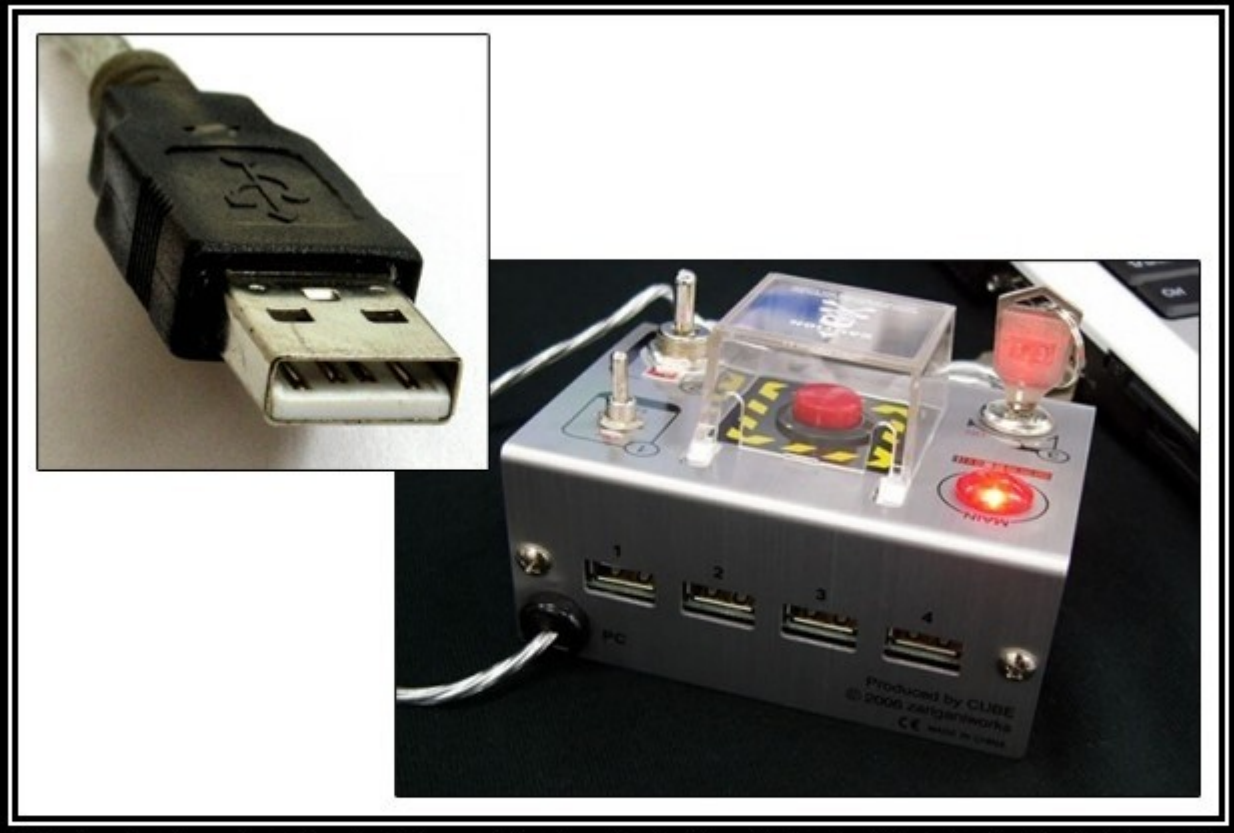
GeeCON
Let's move the Java world!

LSP (by example)

- No, behavior is different
- If I pass a Square to a Rectangle aware function, then this may fail as it may assume that width and heigth are managed separately

- Geometry != Code

GeeCON
Let's move the Java world!



INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

Interface Segregation Principle

- fat classes may happen :(
- usually there are many clients each using a subset of the methods of such classes
- such client classes depend upon things they don't use
  - what happens when the big class changes? all depending modules must also change

GeeCON
Let's move the Java world!

ISP

- ISP states that clients should not know about fat classes
- instead they should rely on clean cohesive interfaces
- you don't want to depend upon something you don't use

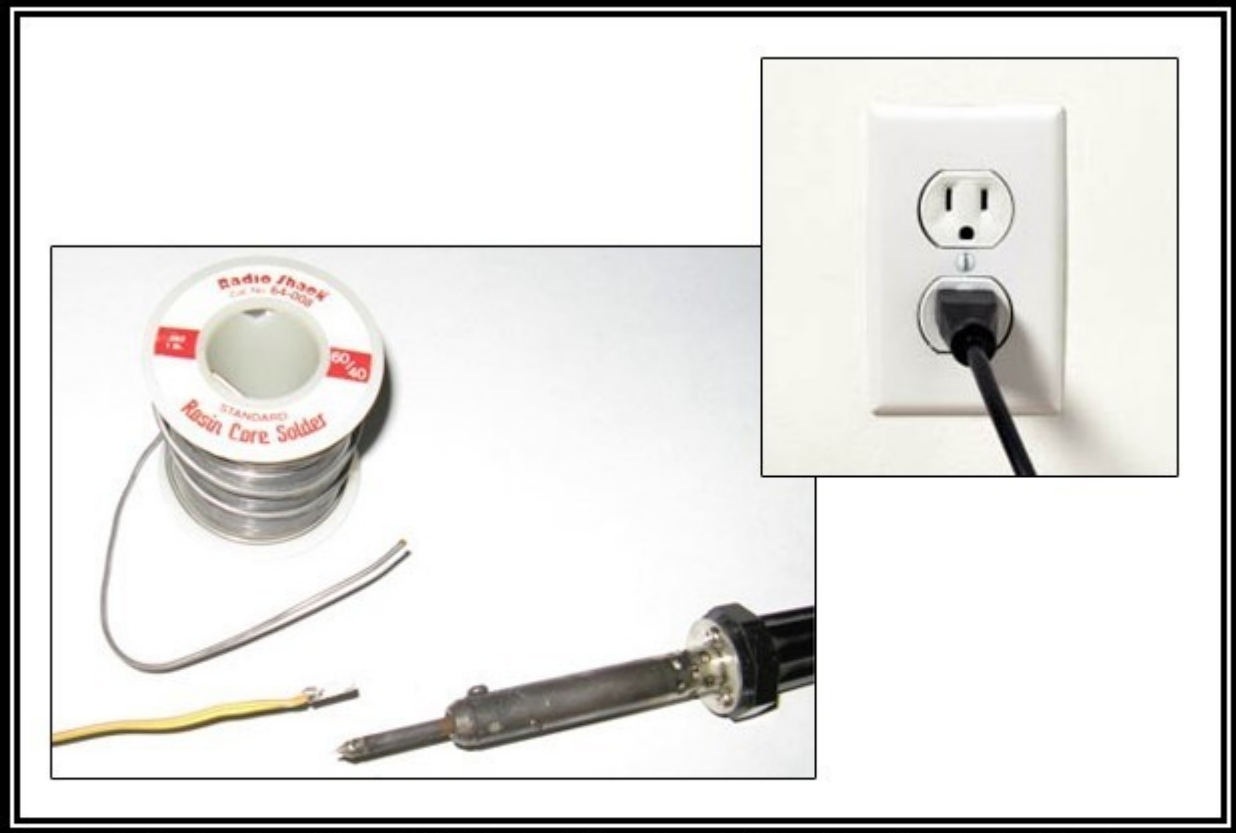
BREAK!

- What is the best comment in source code you have ever encountered? (part 3)

```
// I'm sorry
```

(source: stackoverflow.com)

GeeCON
Let's move the Java world!



DEPENDENCY INVERSION PRINCIPLE

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

Dependency Inversion Principle

- **High level modules should not depend upon low level modules, both should depend upon abstractions**
- **Abstractions should not depend upon details, details should depend upon abstractions**

GeeCON
Let's move the Java world!

DIP?

```
enum OutputDevice {printer, disk};

void copy(OutputDevice dev)
{
    int c;
    while((c=readKeyboard()) != EOF)
    {
        if (dev == printer)
            writePrinter(c);
        else
            writeDisk(c);
    }
}
```

DIP!

```
void copy(Reader input, Writer output)
{
    int c;
    while((c=input.read()) != EOF) {
        output.write(c);
    }
}
```

GeeCON
Let's move the Java world!

DIP

- don't depend on anything concrete, depend only upon abstraction
- high level modules should not be forced to change because of a change in low level / technology layers
- drives you towards low coupling

Conclusion

- **good design** is needed to successfully deal with change
- the main forces driving your design should be **high cohesion** and **low coupling**
- **SOLID** principles put you on the right path

- warning: these principles cannot be applied blindly :)

GeeCON
Let's move the Java world!

Q&A

bruno.bossola@jugtorino.it

GeeCON
Let's move the Java world!