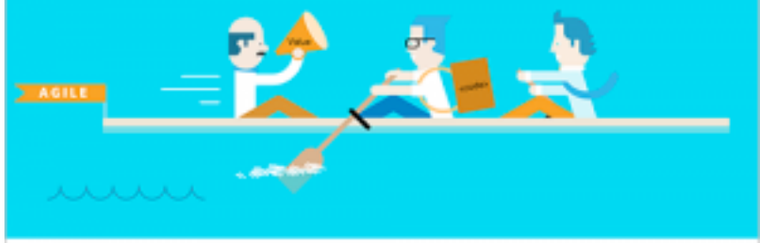


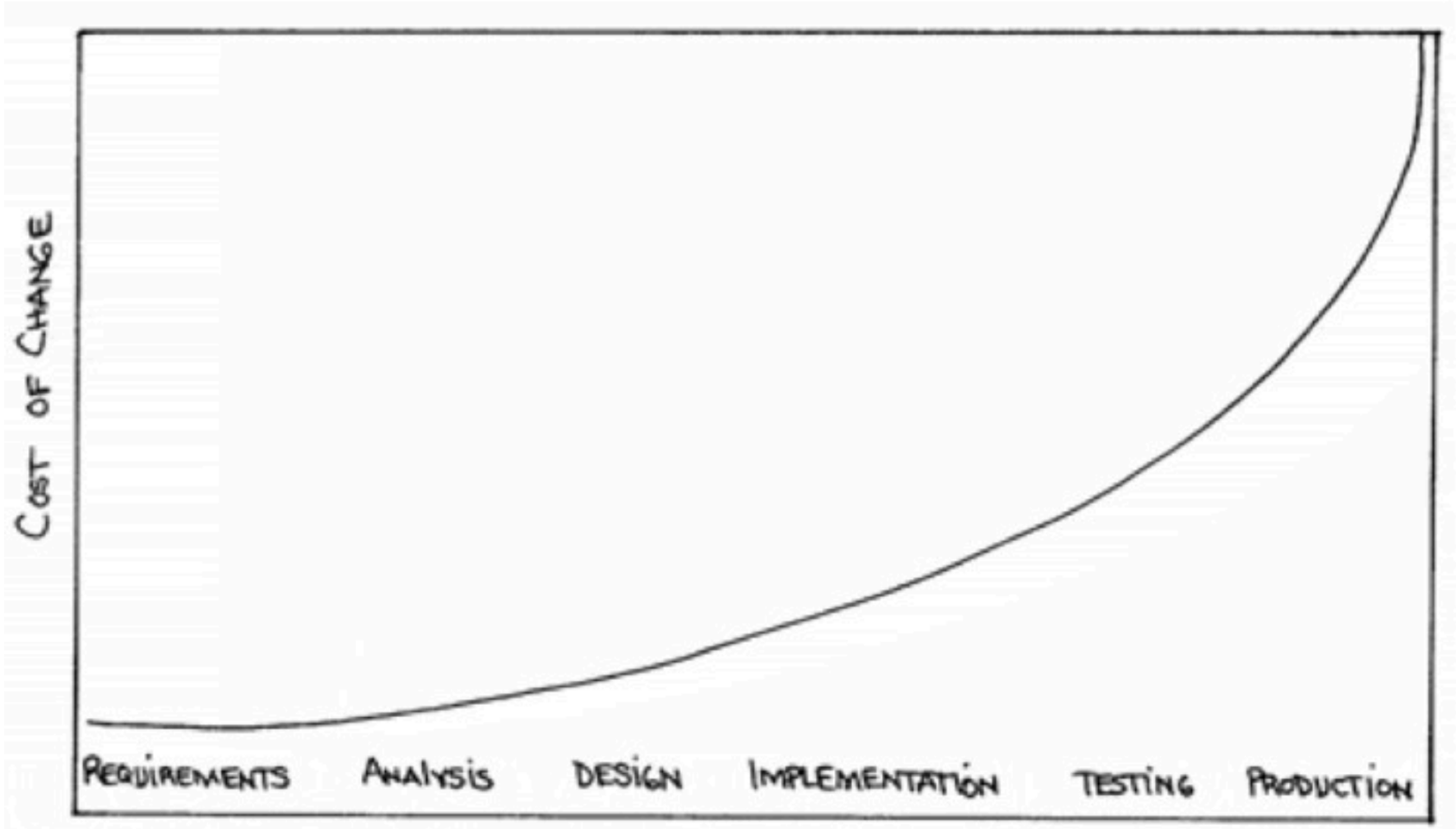
Agilereloaded.it

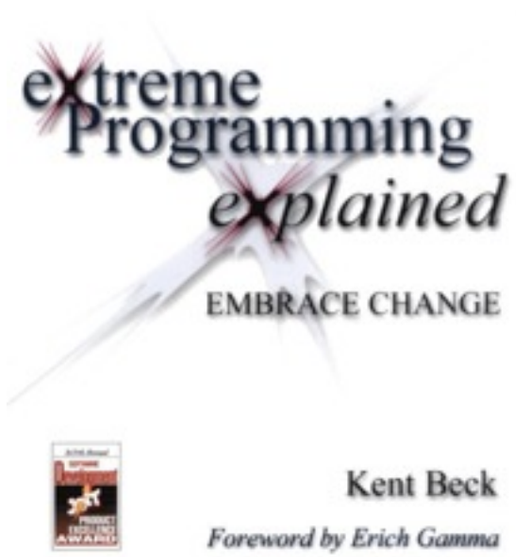
The Open/Closed Principle Kata

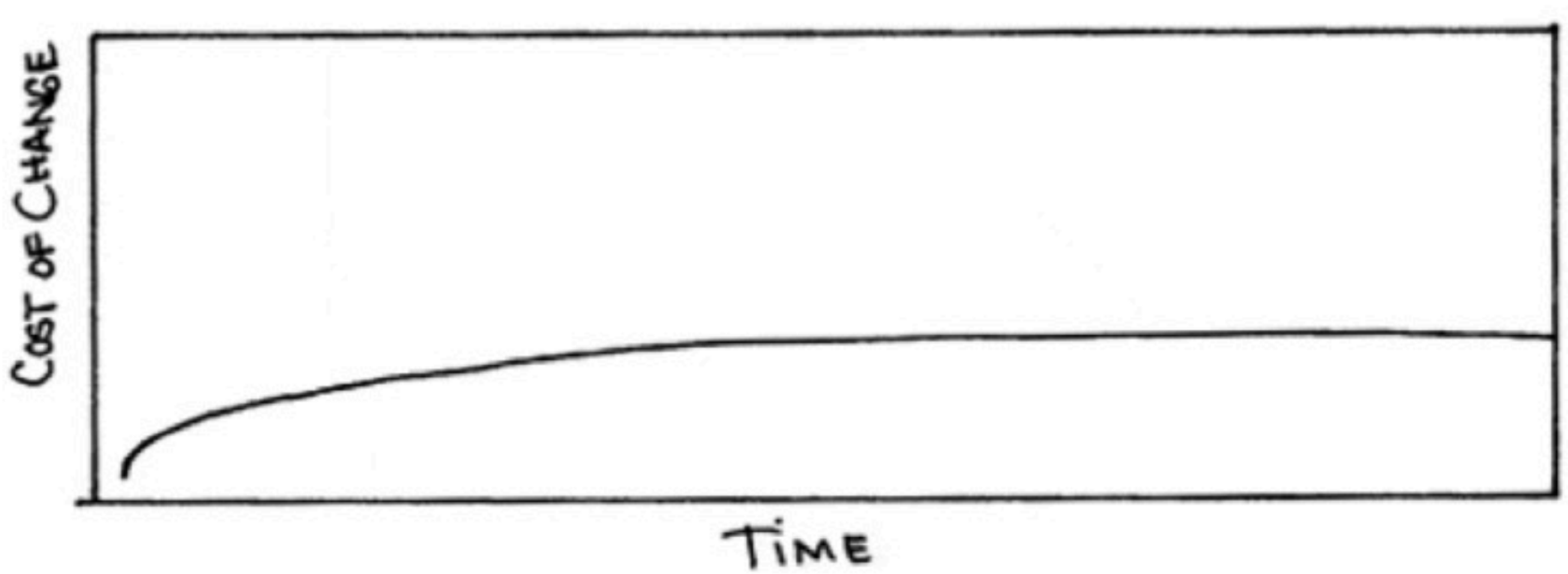
Matteo Vaccari
vaccari@pobox.com
http://matteo.vaccari.name/

~~XP Days Benelux 2010~~
~~Progettazione del Software 2011/12~~
~~Progettazione del Software 2012/13~~
XP2014

Barry Boehm:







From Kent Beck, *XP Explained*

The FizzBuzz Game

1, 2, Fizz!, 4, Buzz!, Fizz!, 7,
8, Fizz!, Buzz!, 11, Fizz!, 13,
14, FizzBuzz!, 16, 17, Fizz!...

If the number is a multiple of 3, say "Fizz"
If it is a multiple of 5, say "Buzz"
If it is a multiple of 3 and 5, say "FizzBuzz"
Otherwise, just say the number.

It's not hard...

```
public String say(Integer n) {
    if (isFizz(n) && isBuzz(n)) {
        return "FizzBuzz";
    }
    if (isFizz(n)) {
        return "Fizz";
    }
    if (isBuzz(n)) {
        return "Buzz";
    }
    return n.toString();
}

public boolean isFizz(Integer n) {
    return 0 == n % 3;
}

// ...
```

New requirement

If it is a multiple of 7, say "Bang"

No problem!

```
public String say(Integer n) {
    if (isBang(n)) {
        return "Bang";
    }
    if (isFizz(n) && isBuzz(n)) {
        return "FizzBuzz";
    }
    if (isFizz(n)) {
        return "Fizz";
    }
    if (isBuzz(n)) {
        return "Buzz";
    }
    return n.toString();
}
```

Wait, that's not what I meant!

If it is a multiple of 3 and 7, say "FizzBang"
If it is a multiple of 5 and 7, say "BuzzBang"
If it is a multiple of 3, 5 and 7, say "FizzBuzzBang"

Hmmm....

```
public String say(Integer n) {
    if (isFizz(n) && isBuzz(n) && isBang(n)) {
        return "FizzBuzzBang";
    }
    if (isBang(n) && isBuzz(n)) {
        return "BuzzBang";
    }
    if (isBang(n) && isFizz(n)) {
        return "FizzBang";
    }
    if (isBang(n)) {
        return "Bang";
    }
    if (isFizz(n) && isBuzz(n)) {
        return "FizzBuzz";
    }
    if (isFizz(n)) {
        return "Fizz";
    }
    if (isBuzz(n)) {
        return "Buzz";
    }
    return n.toString();
}
```

Not so simple any more!

OK. Nobody told you before but...

Adding IFs is evil.

What is refactoring?

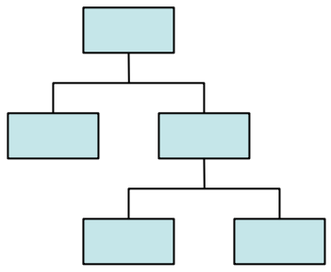
When implementing a program feature, the programmers always ask if there is *a way of changing the existing program to make adding the feature simple*.

After they have added a feature, the programmers ask if they now can see how to make the program simpler

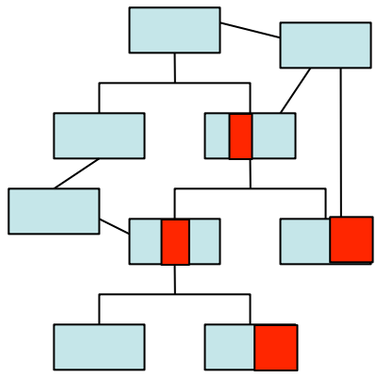
Kent Beck, *XP Explained*

When do we refactor?

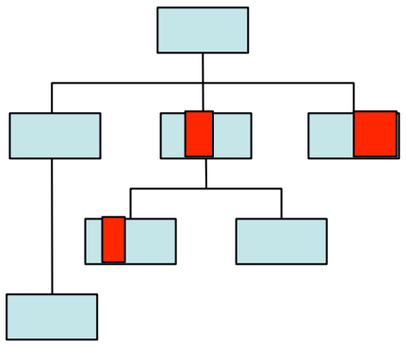
Refactor after



Starting code base

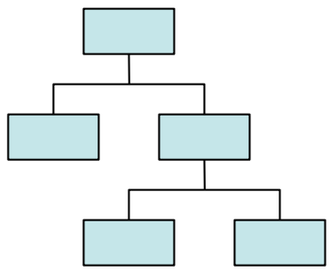


Changes implemented
red == code changed

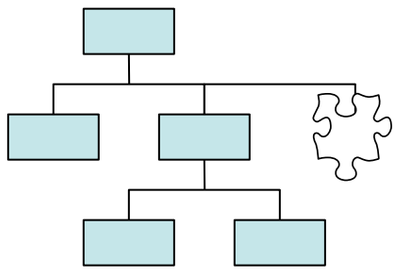


(Hopefully) Code cleaned up

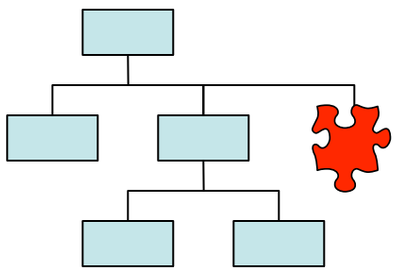
Refactor before



Starting code base



Change design to make room for new feature



Implement feature

From a slide by Dave Nicolette

The Open/Closed Principle

Software entities
(classes, modules, functions, etc.)
should be *open for extension*, but
closed for modification

Bertrand Meyer 1990

New requirements should be
implemented by adding *new files*, not by
changing existing files

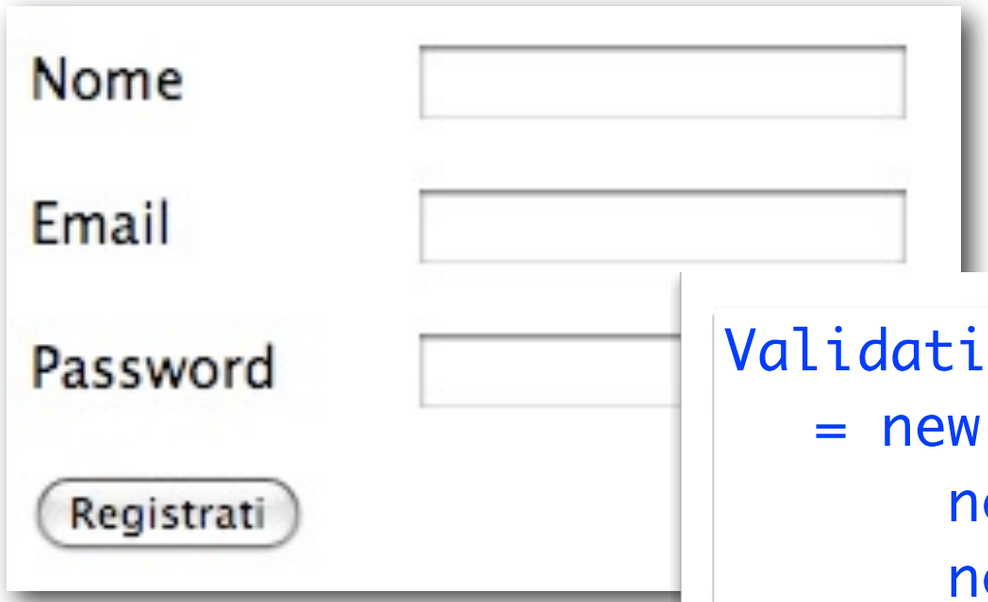
My formulation

When I must add functionality:

- Can I do it by changing *only construction code* and creating *new classes*?
- If I can, I rock! ➪ €€€€
- If I can't, I *refactor* until I can


```
new RegistrationForm();
```

```
class RegistrationForm {
    // ...
    void validate() throws ValidationException {
        if (nameField.value().isEmpty()) {
            throw new ValidationException("Manca il nome");
        }
        if (emailField.value().isEmpty()) {
            throw new ValidationException("Manca l'email");
        }
        // ...
    }
}
```




```
ValidationRuleList validationRules
   = new ValidationRuleList(
        new ValidatePresenceOf("name"),
        new ValidatePresenceOf("email"),
        new ValidateLengthOf("password", 8),
        // ...
      );
new RegistrationForm(validationRules);
```

```
class RegistrationForm {
    // ...
    void validate() throws ValidationException {
        for (ValidationRule rule: validationRules) {
            rule.validate(this);
        }
    }
}
```

Rules for the OCP kata

1. Write a failing test
2. Write a setup that builds an object that makes the test pass
3. Write next failing test
4. Can you make it pass by changing the setup and creating new classes?
   - Yes: great! go back to step 3
   - No: refactor until you can

Refactoring should bring the system in a state where it's possible to implement the next test just by composing objects in the setup method

No new functionality! Current test should still fail

First test: Say the number

Just say the number

say(1) returns "1"

say(2) returns "2"

Second test: Say "Fizz"

When a number is a multiple of 3,
say "Fizz"

say(3) returns "Fizz"
say(6) returns "Fizz"

Third test: say "Buzz"

When a number is a multiple of 5,
say "Buzz"

say(5) returns "Buzz"
say(10) returns "Buzz"

Fourth test: say "FizzBuzz"

When a number is a multiple of 3 and 5, say "FizzBuzz"

say(3*5) returns "FizzBuzz"

Fifth test: say Bang

When a number is a multiple of 7, say "Bang"

say(7) returns "Bang"
say(14) returns "Bang"

Sixth, Seventh, Eighth test:
say FizzBang, BuzzBang, FizzBuzzBang

say(3*7) returns "FizzBang"
say(5*7) returns "BuzzBang"
say(3*5*7) returns "FizzBuzzBang"

Want to know more?

Google for “OCP kata”