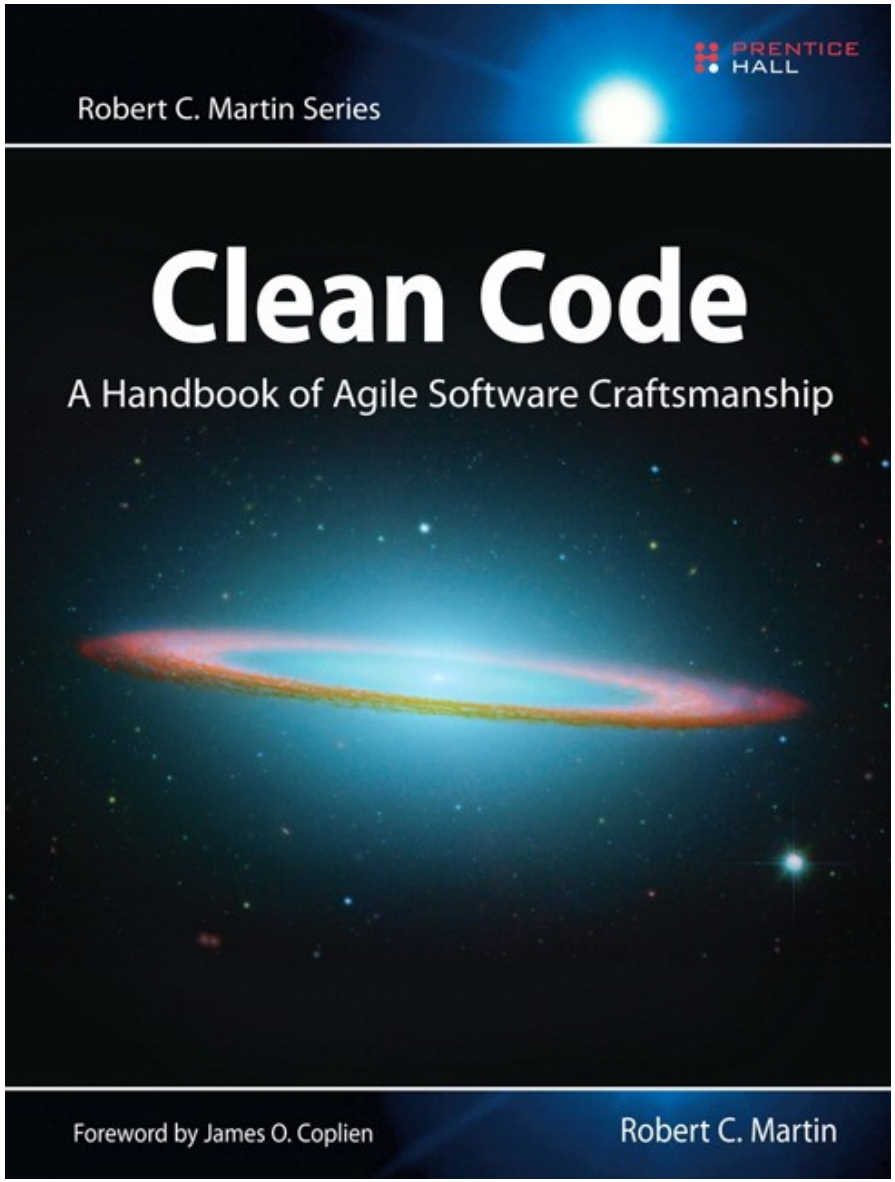
Alpes JUG le lundi 13 décembre

La chasse aux 7 péchés capitaux peut commencer

Par Freddy Mallet
freddy.mallet@sonarsource.com

Pour (re)trouver le droit chemin



Les 7 péchés capitaux

Péchés

Dette technique

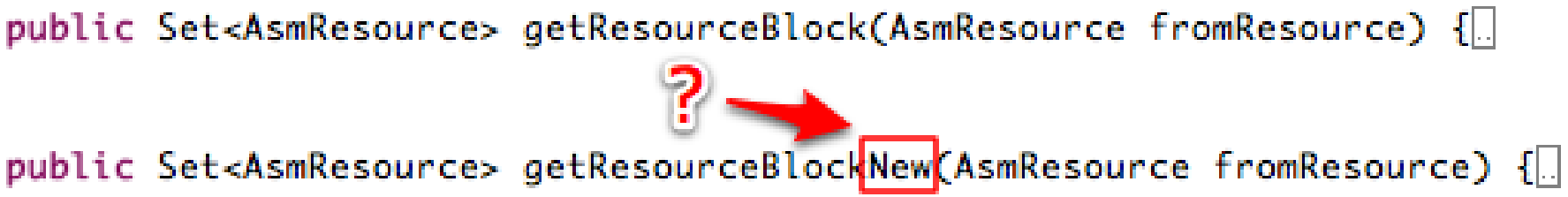
Les 7 péchés capitaux ?

Appliqués au code source

- **Duplications**
- Mauvaise distribution de la complexité
- Mauvais Design
- Pas de tests unitaires
- Pas de respect des standards
- Bugs potentiels
- Pas ou trop de commentaire

Code dupliqué

- Que choisir entre la peste et le choléra ?



Once and only once (Kent Beck)

Tout code dupliqué est une opportunité pour élever le niveau d'abstraction et étoffer la richesse du design

Avant tout une histoire de bon sens

- Avez-vous deux processus pour
  - Passer en production une nouvelle version applicative ?
  - Soumettre un appel d'offre ?
  - Recruter un collaborateur ?

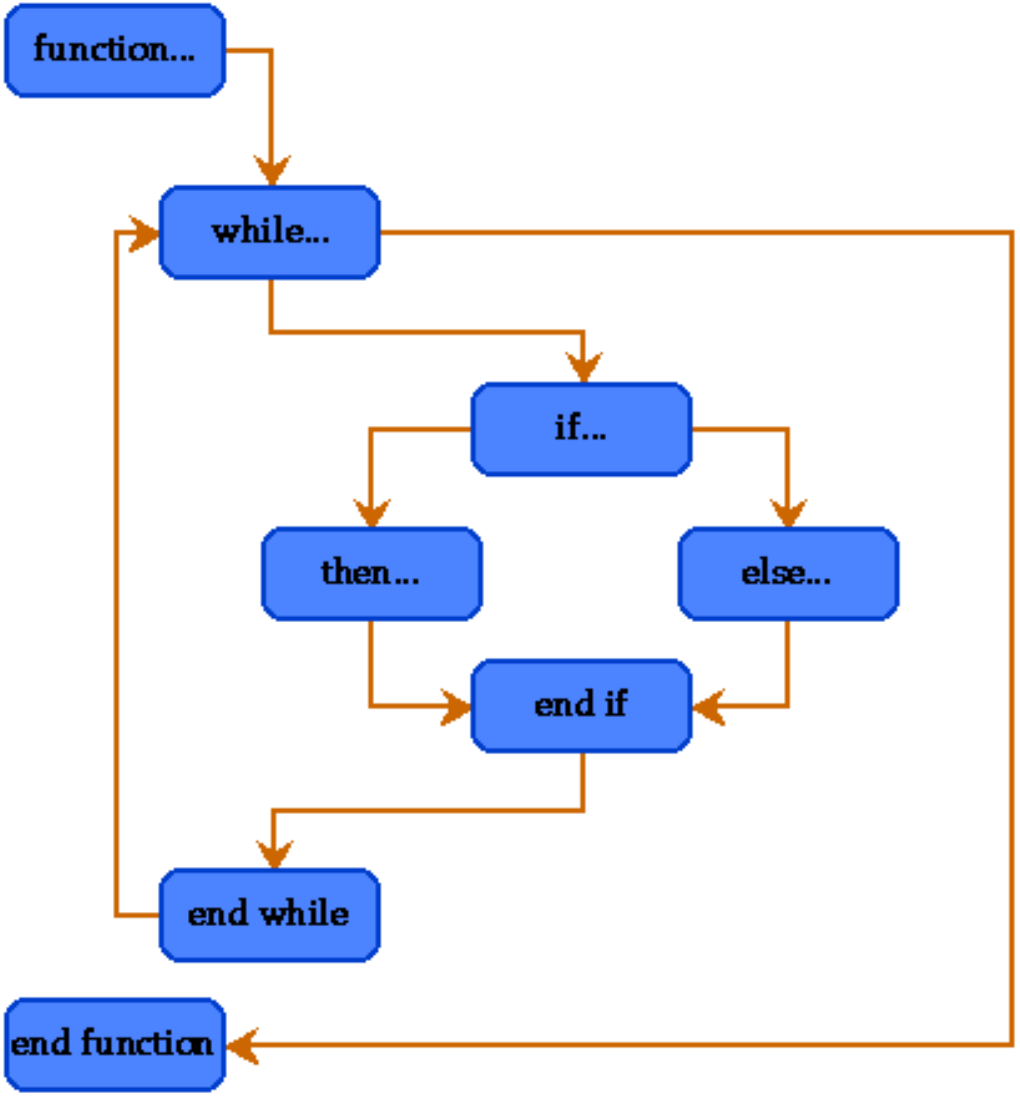
Les 7 péchés capitaux ?

Appliqués au code source

- Duplications
- **Mauvaise distribution de la complexité**
- Mauvais Design
- Pas de tests unitaires
- Pas de respect des standards
- Bugs potentiels
- Pas ou trop de commentaire

Mauvaise distribution de la complexité

Qu'est-ce que la complexité cyclomatique ?



Mauvaise distribution de la complexité

- Vaut-il mieux :
  - 1 méthode d'une complexité de 30
  - 10 méthodes d'une complexité de 3

```
if (size > 0) {
    Object otherValue = null;
    switch (size) {  // drop through
        case 3:
            if (other.containsKey(key3) == false) {
                return false;
            }
            otherValue = other.get(key3);
            if (value3 == null ? otherValue != null : !value3.equals(otherValue)) {
                return false;
            }
        case 2:
            if (other.containsKey(key2) == false) {
                return false;
            }
            otherValue = other.get(key2);
            if (value2 == null ? otherValue != null : !value2.equals(otherValue)) {
                return false;
            }
```

La complexité se gère à tous les étages

- Méthode
- Classe
- Package
- Module

Règle d'or :
Une méthode ou une classe n'est jamais trop petite

La complexité et le SRP

SRP : Single Responsibility Principle

- Une classe ou une méthode doit avoir une et une seule raison de changer
- Si ce n'est pas le cas, il faut diviser (par abstraction ou dissociation) pour mieux régner

Les 7 péchés capitaux ?

Appliqués au code source

- Duplications
- Mauvaise distribution de la complexité
- **Mauvais Design**
- Pas de tests unitaires
- Pas de respect des standards
- Bugs potentiels
- Pas ou trop de commentaire

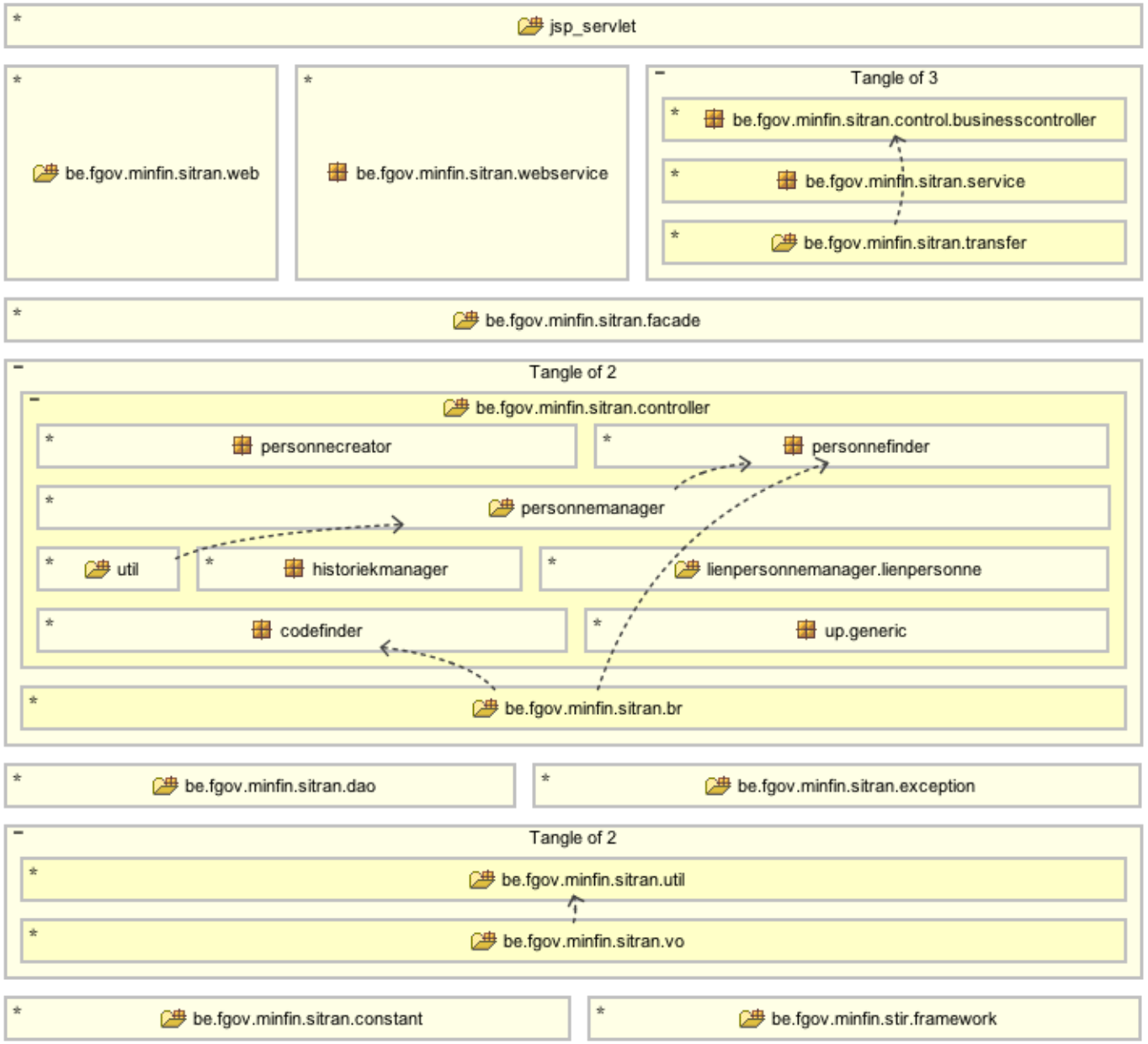
Mauvais design

Comportant deux niveaux d'abstraction

- Quel package/classe est responsable de quoi ?

Les couches d'architecture

Les cycles apparaissent comme le nez au milieu de figure



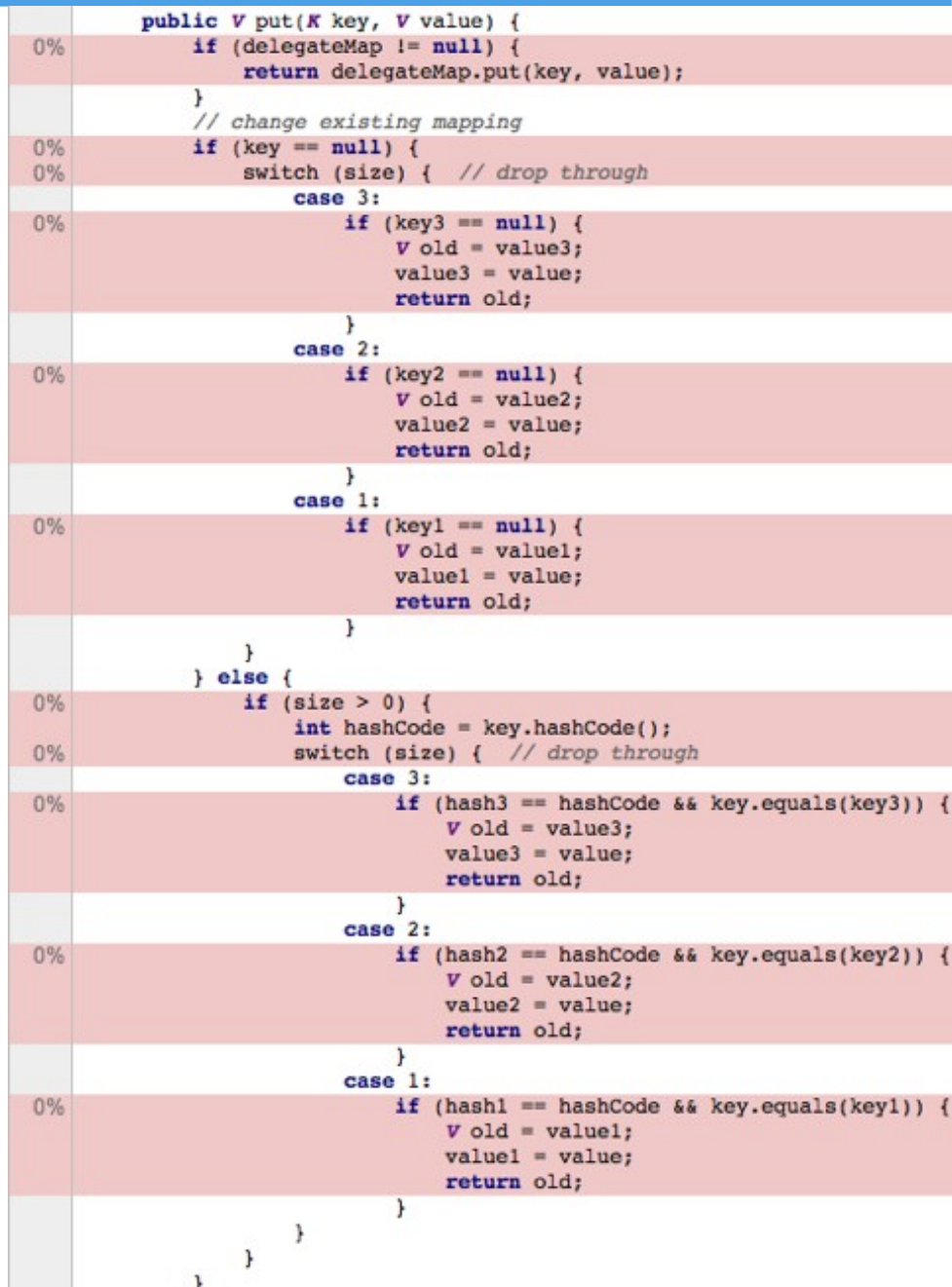
Les 7 péchés capitaux ?

Appliqués au code source

- Duplications
- Mauvaise distribution de la complexité
- Mauvais Design
- **Pas de tests unitaires**
- Pas de respect des standards
- Bugs potentiels
- Pas ou trop de commentaire

Peu ou pas d'utilisation des tests unitaires

- Merci d'ajouter un nouveau cas et d'éviter toute régression bien évidemment...

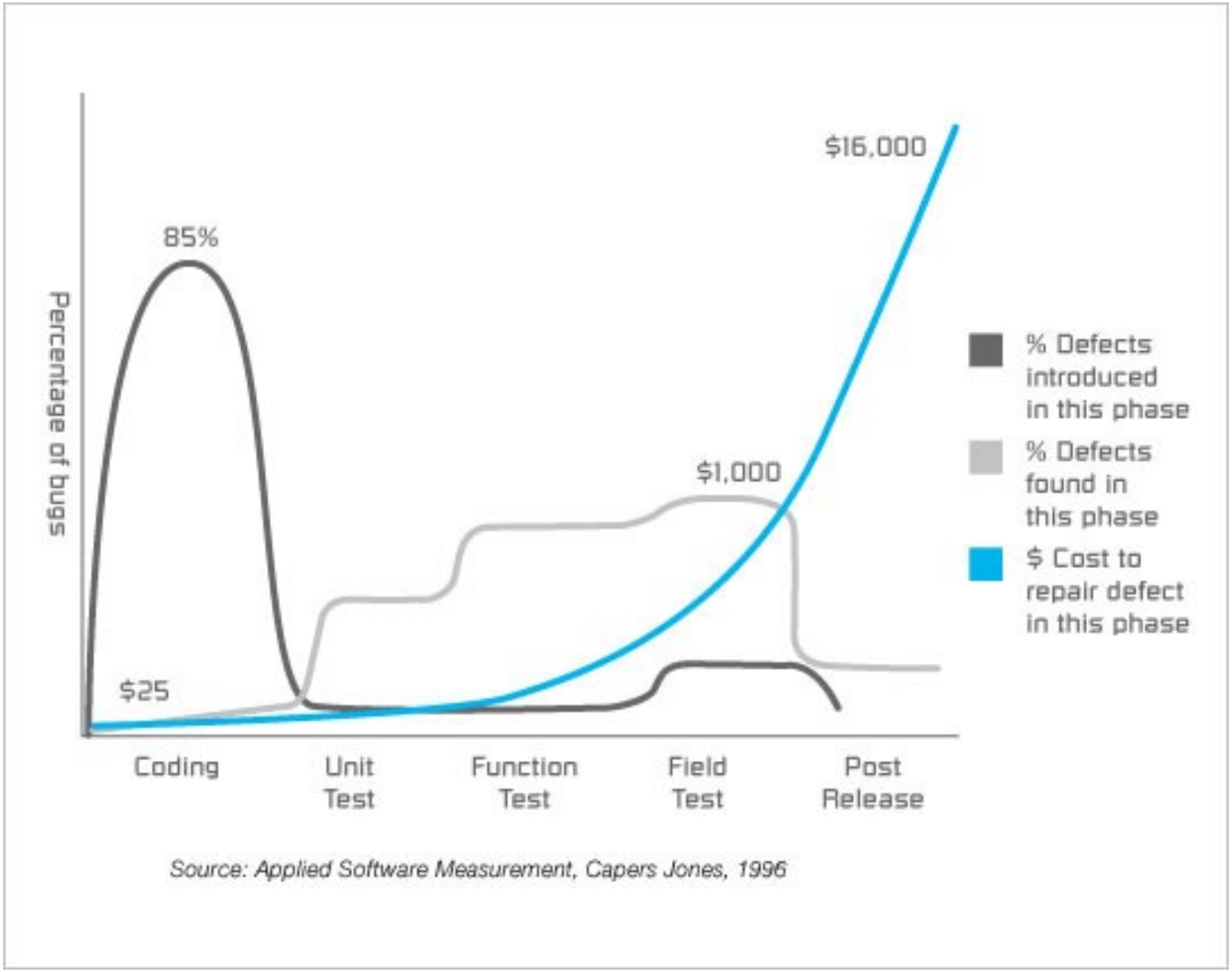


```
    public V put(K key, V value) {
0%      if (delegateMap != null) {
            return delegateMap.put(key, value);
        }
        // change existing mapping
0%      if (key == null) {
0%          switch (size) {  // drop through
                case 3:
0%                  if (key3 == null) {
                        V old = value3;
                        value3 = value;
                        return old;
                    }
                case 2:
0%                  if (key2 == null) {
                        V old = value2;
                        value2 = value;
                        return old;
                    }
                case 1:
0%                  if (key1 == null) {
                        V old = value1;
                        value1 = value;
                        return old;
                    }
            }
        } else {
0%          if (size > 0) {
                int hashCode = key.hashCode();
0%              switch (size) {  // drop through
                    case 3:
0%                      if (hash3 == hashCode && key.equals(key3)) {
                            V old = value3;
                            value3 = value;
                            return old;
                        }
                    case 2:
0%                      if (hash2 == hashCode && key.equals(key2)) {
                            V old = value2;
                            value2 = value;
                            return old;
                        }
                    case 1:
0%                      if (hash1 == hashCode && key.equals(key1)) {
                            V old = value1;
                            value1 = value;
                            return old;
                        }
                }
            }
        }
```

Un code sans tests unitaires est soit

- Du code jetable
- Du code dont le coût d'ajout d'une fonctionnalité et des régressions associées est sans importance
- Du code d'une pure application CRUD (Create Read Update Delete)
- Du code legacy dont la piètre qualité entraine un coût d'écriture et de maintenance des tests unitaires trop élevé

Les tests : avant tout une histoire de ROI



Source: Applied Software Measurement, Capers Jones, 1996

L'effet boule de neige du TDD

- Raisonnement en terme de contrat
  - Donc en terme de responsabilité (SRP)
  - Expressivité du vocabulaire
- Approche itérative et incrémentale au niveau le plus fin
  - Méthodes peu complexes
  - Linéarité des développements et donc meilleur prédictibilité des délais

Les 7 péchés capitaux ?

Appliqués au code source

- Duplications
- Mauvaise distribution de la complexité
- Mauvais Design
- Pas de tests unitaires
- **Pas de respect des standards**
- Bugs potentiels
- Pas ou trop de commentaire

Non respect des standards

La gestion des exceptions

```
try {
   computeOrder(order);
} catch (RuntimeException e) {
   System.out.println("The order can't be computed!");
}
```

Nombre d'arguments d'une fonction

- L'expressivité et la testabilité d'une fonction diminue exponentiellement à chaque ajout d'un argument
- A partir de trois arguments, il est temps de se remettre en question

```
assertEquals(java.lang.String message, java.lang.Object expected, java.lang.Object actual)
```

Asserts that two objects are equal.

```
assertThat(x, is(3));
assertThat(x, is(not(4)));
```

Taille verticale et horizontale d'un fichier

Quel traitement réservez-vous aux emails qui ne tiennent pas sur un écran ?

Règle d'indentation et de formatage

Il est aussi aisé de se mettre à lire du code suivant un nouveau standard que de comprendre les panneaux de signalisation quand on débarque dans un nouveau pays

Les 7 péchés capitaux ?

Appliqués au code source

- Duplications
- Mauvaise distribution de la complexité
- Mauvais Design
- Pas de tests unitaires
- Pas de respect des standards
- **Bugs potentiels**
- Pas ou trop de commentaire

Bugs potentiels

```
if (listeners == null)
    listeners.remove(listener);
```

Sun java : JDK1.6.0, b105,
sun.awt.x11.XMSelection
lines 243-244

Exemple

Valeur de retour ignorée

```
String aString = "bob";
b.replace('b', 'p');
if(b.equals("pop"))
```

Exemple, suite

Null pointer

```
Person person = aMap.get("bob");
if (person != null) {
    person.updateAccessTime();
}
String name = person.getName();
```

Les 7 péchés capitaux ?

Appliqués au code source

- Duplications
- Mauvaise distribution de la complexité
- Mauvais Design
- Pas de tests unitaires
- Pas de respect des standards
- Bugs potentiels
- **Pas ou trop de commentaires**

Pas ou trop de commentaires


```
/**
 * blabla
 * blabla
 * blabla
*/
public void validate(String param1, String param2
                     String param3, String param4,
                       int param5) {...}

public boolean isValid(Order order) {...}
```

Expressivité du vocabulaire

Versus densité des commentaires

```
public List<int[]> getThem(){
  List<int[]> list1 = new ArrayList<int[]>();
  for(int[] x : theList){
    if(x[0] == 4){
      list1.add(x);
    }
  }
  return list1;
}
```

Avant

```
public List<Cell> getFlaggedCells(){
  List<Cell> flaggedCells = new ArrayList<Cell>();
  for(Cell cell : gameBoard){
    if(cell.isFlagged()){
      flaggedCells.add(cell);
    }
  }
  return flaggedCells;
}
```

Après

La javadoc : un commentaire particulier

- L'utilisation doit être limitée aux API
- La Javadoc s'adresse en priorité à des consommateurs et non aux équipes en charge de la maintenance évolutive

Un commentaire doit être utile

Ou ne pas être

- Pour amplifier l'importance d'une logique

```
String listItemContent = match.group(3).trim();
// the trim is really important. It removes the starting
// spaces that could cause the item to be recognized
// as another list
```

- Pour apporter une vision dynamique

A typical invocation sequence is thus

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

- Quelques anti-patterns

```
i++; // increment i
```

```
// 04-Sep-2003 - Implemented Comparable. Updated the isInRange javadocs
// 05-Jan-2005 - Fixed ug in addYears() method
```

Les 7 péchés capitaux ?

Appliqués au code source

- Duplications
- Mauvaise distribution de la complexité
- Mauvais Design
- Pas de tests unitaires
- Pas de respect des standards
- Bugs potentiels
- Pas ou trop de commentaires

Questions & Réponses

Merci

http://sonar.codehaus.org
http://www.sonarsource.com

sonar