



08

Chapitre

Refonte de code (Refactoring)

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

« «Si déboguer c'est supprimer des bugs,
alors programmer ne peut être que de les ajouter. »
Edsger Dijkstra

Objectif du chapitre

- Présentation de quelques règles de refonte du code pour en améliorer sa qualité.
- À l'issue de ce chapitre :
 - Vous serez sensibilisé à la malléabilité du code.
 - Vous serez en mesure de lutter contre le pourrissement du code (*rotten code*).
 - Vous saurez refondre du code pourri.

Plan du chapitre

1

Nécessité de la
refonte
de code.

2

Quelques
exemples de
refonte
de code.

Qu'est ce que la refonte de code ?

4

■ Définition

- La refonte de code est le processus qui consiste à changer le code d'un système logiciel de telle manière qu'il n'altère pas le comportement extérieur du code tout en améliorant sa structure interne.

■ Mise en œuvre

- Il requière la présence de tests (exécutables de préférence).
- Sa réalisation nécessite l'utilisation d'environnements de développement appropriés (menu « *refactor* »).
- Il va de pair avec la notion de « code propre ».

Qu'est ce que n'est pas la refonte de code

5

- Conception
 - Il ne correspond pas à l'introduction de nouvelles fonctionnalités.
- Optimisation
 - Il ne correspond pas non plus à l'optimisation des performances qui conduit souvent à un code qui devient difficile à comprendre.

Pourquoi faire de la refonte de code ?

6

■ Raisons

- Pour ne pas accumuler de **dette technique**.
- Pour améliorer la logique de conception du logiciel.
- Pour rendre le système plus simple à comprendre et donc à maintenir, étendre et vérifier.
- Pour aider à trouver les bugs.

■ Principe KISS : « Kept It Simple, Stupid. »

- L'objectif est de toujours garder le code le plus simple possible.
- Attention : faire simple, c'est compliqué et relève généralement d'un processus d'affinage.
- « La perfection n'est atteinte, non pas lorsqu'il n'y a plus rien à ajouter, mais lorsqu'il n'y a plus rien à enlever. » **Antoine de Saint-Exupéry**

Quand faire de la refonte de code ?

7

- Lors du développement.
 - Après l'écriture du code d'une fonctionnalité et de ses tests.
- Pendant la revue de code.
- Lors de la reprise du code.
 - Extension
 - Correction de bug.

Exemple de refonte de code

- "Bad Smells in the code"
 - Duplication de code.
 - Longues méthodes.
 - Grandes classes.
 - Longue liste de paramètres.
 - Nécessité de commentaire.
 - ...

-> Sélection dans le catalogue de Martin Fowler

Factoriser le code redondant

- Vous voyez la même structure de code à plus d'un endroit et portant la même sémantique.
 - Rassembler le code commun dans une méthode à part.
 - Principe DRY : « Don't Repeat Yourself. »

Remplacer un nombre magique par une constante symbolique

- Vous avez un nombre littéral avec un sens particulier.

```
double potentialEnergy( double mass, double height ) {  
    return mass * 9.81 * height;  
}
```



- Créer une constante, la nommer en fonction de son rôle, et remplacer le nombre par cette constante.

```
static final double GRAVITATIONAL_CONSTANT = 9.81;  
double potentialEnergy( double mass, double height ) {  
    return mass * GRAVITATIONAL_CONSTANT * height;  
}
```

Supprimer les doubles négations

- Vous avez une condition avec une double négation.

```
if (!item.isNotFound()) { }
```



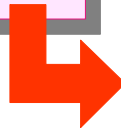
- Rendre cette condition positive.

```
if (item.isFound()) { }
```

Remplacer une méthode par un objet méthode

- Vous avez une longue méthode qui utilise des variables locales

```
public final class Order...  
    float price(int a, Obj o) {  
        double primaryBasePrice;  
        double secondaryBasePrice;  
        double tertiaryBasePrice;  
        // long computation  
    }  
}
```



```
public final class Order..  
    float price() {  
        return new PriceCalculator().compute();  
    }  
    private final class PriceCalculator {  
        private double _primaryBasePrice;  
        private double _secondaryBasePrice;  
        private double _tertiaryBasePrice;  
        protected float compute() {  
            // long computation that uses method2...  
        }  
        private int method2() {  
            ... // it uses global variables  
        }  
    }  
}
```

- Transformer la méthode en une classe interne de telle manière que les variables locales deviennent des attributs de cet objet puis décomposer en sous-méthodes privées (abstraction hypostatique).

Introduire une variable explicative

- Vous avez une expression de condition compliquée.

```
if (platform.indexOf("MAC") > -1
    && browser.indexOf("IE") > -1
    && wasInitialized() && resize > 0) {
    // do something
}
```




- Mettre le résultat de l'expression, ou une partie de cette expression, dans une variable temporaire avec un nom qui explicite son rôle.

```
boolean isMacOs = platform.indexOf("MAC") > -1;
final boolean isIEBrowser = browser.indexOf("IE") > -1;
final boolean wasResized = resize > 0;
if (isMacOs && isIEBrowser && wasInitialized() && wasResized) {
    // do something
}
```

Préserver un objet entier

- Vous utilisez plusieurs valeurs d'un objet et passez ces valeurs comme paramètres d'une méthode.

```
int low = daysTempRange().getLow();  
int high = daysTempRange().getHigh();  
withinPlan = plan.withinRange(low, high);
```



- Envoyer l'objet en entier.

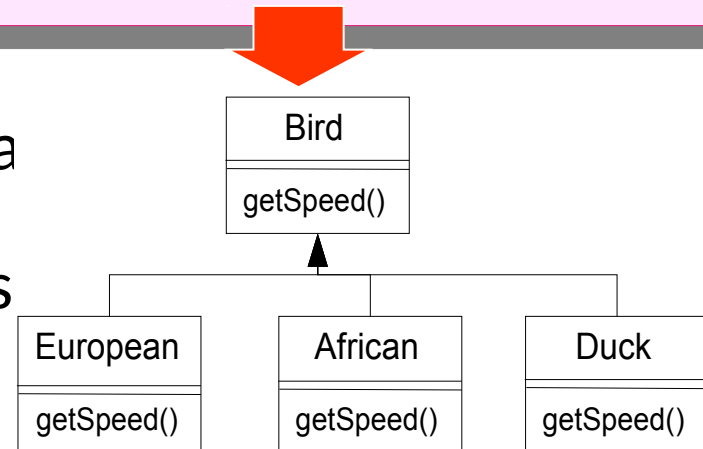
```
withinPlan = plan.withinRange(daysTempRange());
```

Remplacer les conditions par le polymorphisme

- Vous avez une conditionnelle qui sélectionne différents comportements en fonction du type de l'objet.

```
double getSpeed() {  
    switch (_type) {  
        case EUROPEAN: return getBaseSpeed();  
        case AFRICAN: return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;  
        case DUCK: return (_isWood) ? 0 : getBaseSpeed();  
    }  
}
```

- Créer des sous-classes et déplacer chaque partie de la conditionnelle dans une méthode redéfinie dans une sous-classe et utilisez le polymorphisme.



Remplacer un code d'erreur par une exception

- Vous avez une méthode qui retourne un code spécial pour indiquer une erreur.

```
int withdraw( int amount ) {  
    if (amount > _balance) {  
        return -1;  
    }  
    _balance -= amount;  
    return 0;  
}
```



- Lever une exception à la place.

```
void withdraw( int amount ) throws BalanceException {  
    if (amount > _balance) {  
        throw new BalanceException();  
    }  
    _balance -= amount;  
}
```


Remplacer un paramètre par une méthode

17

- Vous avez un objet qui appelle une méthode, et passe le résultat comme paramètre d'une méthode.

```
int basePrice = _quantity * _itemPrice;  
discountLevel = getDiscountLevel();  
double finalPrice = discountedPrice(basePrice, discountLevel);
```



- Supprimer le paramètre et laisser le receveur appeler la méthode.

```
int basePrice = _quantity * _itemPrice;  
double finalPrice = discountedPrice(basePrice);  
// getDiscountLevel() est appelée à l'intérieur de discountedPrice()
```

Introduire un objet paramètre

- Vous avez un groupe de paramètres qui vont naturellement ensemble.

```
amountInvoicedIn(start: Date, end: Date)  
amountReceivedIn(start: Date, end: Date)  
amountOverdueIn(start: Date, end: Date)
```



- Remplacer les paramètres par un objet qui les rassemble.

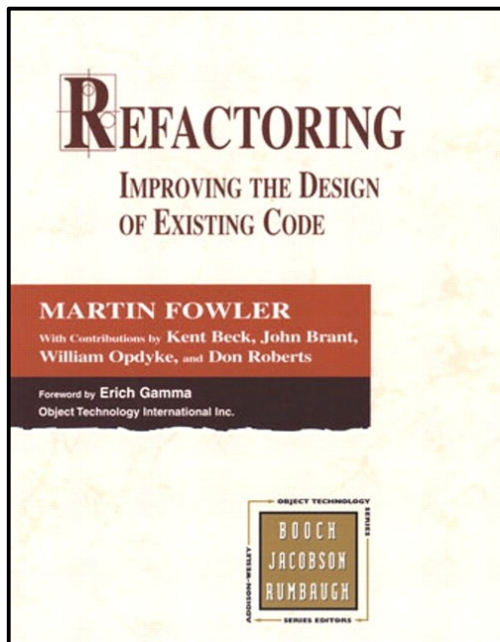
```
amountInvoicedIn(DateRange)  
amountReceivedIn(DateRange)  
amountOverdueIn(DateRange)
```

Que retenir de ce chapitre ?

- Le refactoring de code permet d'améliorer la structure interne du code.
- Il ne peut pas se faire sereinement sans tests dynamiques qui permettent de se prémunir contre la régression.
- Le refactoring s'inscrit après le codage d'une fonctionnalité et de ses tests.
- N'oubliez pas la règle des scouts :
« Laissez le campement plus propre que vous l'avez trouvé en arrivant »

Lectures

- Martin Fowler, Refactoring « *Improving the Design of Existing Code* », Addison-Wesley Professional, 1999.



- Martin Fowler, « *Refactoring Home Page* », <http://refactoring.com/catalog/index.html>