ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

07
Chapitre

Refonte de code (Refactoring)

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

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

Qu'est ce que la refonte de code ?

- **Définition**
  - La refonte de code est le processus qui consiste à changer le code d'un système logiciel :
    - Pour **améliorer la structure interne** du code
    - Sans **altérer le comportement extérieur** du code
  - La refonte de code nécessite des tests automatiques (voir chapitre 8) pour s’assurer que l’on ne change pas le comportement
- Activité fondamentale de l’artisan du code
  - On n’écrit pas du code propre en une seule fois. Il faut retravailler sans cesse le code

Pourquoi faire de la refonte de code ?

- Pour ne pas accumuler de **dette technique**
- Pour améliorer la logique interne du logiciel
- Pour rendre le système plus simple à comprendre et donc à maintenir, étendre et vérifier
  - Principe KISS : « Keep It Simple, Stupid. »
    - « 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
- Pour aider à trouver les bugs (p. ex. les Schroedinbugs)

Quand faire de la refonte de code ?

1/ Lors du développement d'une fonctionnalité

- Induit une nouvelle façon de coder :
  1) Vite fait, Pas propre : *Quick and Dirty*
  2) Écriture des tests
  3) Refonte du code : *Refactoring*

Quand faire de la refonte de code ?

2/ Lors de la maintenance corrective ou évolutive

- Pendant la correction du code bugué ou l'insertion du nouveau code, en profiter pour améliorer le code existant
  - Application de la règle de boy-scouts : « *Laissez le campement (code) plus propre que vous l'avez trouvé en arrivant (en l'éditant)* »

Exemples de refonte de code

- Des indices d'une refonte nécessaire : "*bad smells in the code*"
  - Duplication de code
  - Longue méthode (*brain method*)
  - Grande classe (*god class*)
  - Longue liste de paramètres
  - Nécessité d'ajouter un commentaire
  - …

Sélection d’exemples tirés du catalogue de Martin Fowler

http://refactoring.com/catalog/index.html

Remplacer un nombre magique par une constante symbolique

- Vous avez un nombre avec un sens particulier (*Magic number*)

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

```
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()) { }
```

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

Supprimer les arguments conditionnels

- Les arguments booléens compliquent la signature, on ne sait pas à quoi correspond l'argument sans aller voir le code
  - Booking book(customer, true);

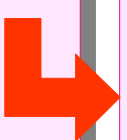
```
class Hotel { ...
   public Booking book(Customer aCustomer, boolean isPremium) {
    if (isPremium) {
       // Logique pour une réservation premium
    } else {
       // Logique pour une réservation classique
    }
}
```

```
class Hotel { ...
   public Booking regularBook(Customer aCustomer);
   public Booking premiumBook(Customer aCustomer);
}
```

Remplacer une méthode par un objet méthode

- Vous avez une longue méthode qui utilise des variables locales

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



```
public class Order..
 float price() {
  return new PriceCalculator.compute();
 }

class PriceCalculator {
   private static double _primaryBasePrice;
   private static double _secondaryBasePrice;
   private static double _tertiaryBasePrice;
   protected static float compute() {
     // long code avec method1, method2
   }
   private int method1() {
     ... // utilise des variables globales
   }
   private int method2() {
     ... // utilise des variables globales
   }
```

Introduire une variable explicative

- Vous avez une expression de condition compliquée

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

```
final boolean isMacOs = platform.indexOf("MAC") > -1;
final boolean isSafariBrowser = browser.name.indexOf("Safari") > -1;
final boolean wasResized = resize > 0;
if (isMacOs && isSafariBrowser && 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);
```

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

Introduire un objet paramètre

- Vous avez un groupe de paramètres qui vont naturellement ensemble
  - Créer une classe spécifique (*ne pas hésiter à faire des classes*)

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

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

class DateRange {
   Date start;
   Date end;
}
```

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;
}
```

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

Attention à ne pas faire de Refuctering

- Refuctoring : processus qui consiste à prendre un morceau de code et, grâce à une série de petites modifications irréversibles, à le rendre totalement impossible à maintenir par quiconque sous prétexte d'optimisation

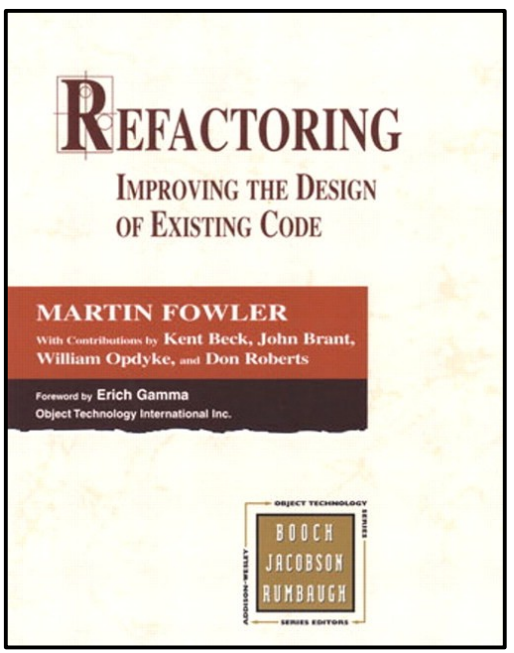
Mise en œuvre

Atelier de l’artisan du logiciel

- La refonte nécessite l'utilisation d'IDE
  - Menu « *refactor* » dans *IntelliJ IDEA* et *Visual Studio Code*

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

Démo refonte de code
Kata Ératosthène