Génie Logiciel : Conception

Ré-usinage Logiciel (aka Refactoring)

Sébastien Mosser
INF-5153, Hiver 2019, Cours #11.1
UQÀM

"Refactoring"

A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior

"To refactor"

To restructure software by applying a series of refactorings without changing its observable behavior

Refactoriser ?
Refactorer ?

Définition
Opération qui consiste à retravailler le code source, non pas pour ajouter une fonctionnalité supplémentaire au logiciel, mais pour améliorer sa lisibilité et simplifier sa maintenance.

Terme privilégié

réusinage n. m.

Terme déconseillé

refactorisation

L'expression *refactorisation*, calquée sur l'anglais *refactoring*, est un faux ami. En effet, l'origine du mot anglais *refactoring* est *factory* (*usine*). Cela n'a rien à voir avec le terme français *factorisation* (ou son dérivé *refactorisation*) qui, en algèbre, est associé à l'utilisation d'un produit de facteurs.

Refactoring versus Over-engineering

- Refactoring :
  - Transformations dans le code exploitant ce que l'on à appris du passé
- Over-engineering
  - Transformation dans le code exploitant des spéculations sur le futur

- Comment mesurer la qualité d'une spéculation ?

Role des Tests

A change made to the internal structure of software to make it easier to understand and cheaper to modify **without changing its observable behavior**

Quelle rapport avec la conception ?

- Le réusinage est un complément à la conception amont
  - Rappel : la conception parfaite n'existe pas, on cherche à tendre vers la moins pire
- Le travail d'analyse / conception amont est un travail d'anticipation
- Le réusinage permet de réparer ce qui n'as pas été vu lors de la conception
  - Mais entre temps on à livré de la valeur

- Principe de l'architecture émergente
  - Requiert de l'experience, et n'est pas adapté à toutes les équipes.

Pourquoi ré-usiner le code ?

- Les programmes ...
  - ... complexe à lire sont complexe à modifier;
  - ... avec de la duplication dans la logique d'affaire sont complexe à modifier;
  - ... avec une logique d'affaire pleines de conditions sont complexe à modifier;
- ... sont complexe à modifier !
- On définit le concept de "dette technique" (technical debt)
  - Il est normal de s'endetter au démarrage du développement d'un produit
  - Mais si on ne rembourse pas sa dette, on fait faillite (anti-patrons, bad smells)

Rôle du ré-usinage dans le produit

- Le ré-usinage permet :
  - D’améliorer la conception du logiciel au fur et à mesure de son développement;
  - De rendre de le logiciel plus lisible;
  - De trouver des bugs dans le code;
  - De développer plus rapidement par la suite.
- C’est une activité importante dans le cycle de vie d’un logiciel
  - Il ne faut pas la négliger dans vos projets

Quand ré-usiner ? (règle de 3)

The first time you do something, you **just do it**.

The second time you do something similar, you **wince at the duplication**, but you **do the duplicate** thing anyway.

The third time you do something similar, **you refactor**

Quand ré-usiner ? (concrètement)

- Quand vous ajoutez une nouvelle fonction
  - Quitte à toucher le code, autant l’embellir au passage
- Quand vous fixez un bug dans le produit
  - Quitte à nettoyer, autant nettoyer en profondeur
- Quand vous intégrez vos modifications avec celles des autres
  - Par exemple en faisant une revue de code.

Concrètement, c'est quoi un ré-usinage ?

- Peut agir sur le code, à plusieurs niveau
  - Méthode, classe
  - Structure (p.-ex., déplacer des méthodes entre objets)
  - Logique d'affaire
- C'est une transformation iso-fonctionnelle du code
  - Le comportement reste le même
  - On améliore la structure pour la clarifier, et améliorer la conception
- Ne pas mélanger ajout fonctionnel et ré-usinage

Exemples de ré-usinage #1

- **Extract method:**
  - Fragment de code qui peut être sémantiquement regroupé ensemble
  - Transformer ce fragment en une méthode qui porte le nom associé à la sémantique
- **Inline method:**
  - Méthode dont le corps est tout aussi parlant que son appel
  - Remplacer les appels à la méthode par le corps de celle-ci

Comment prendre la décision ?

"Bad smells" versus "Antipatterns"

- **Un anti-patron** est une erreur profonde (et typique) qui à des conséquence grave
  - Il faut les détecter et les réparer pour ne pas mettre en péril le produit
- **Une mauvaise odeur**, c'est moins grave
  - Des erreurs (aussi typique), mais plus légères
  - On peut vivre avec "certaines" mauvaises odeurs
  - Mais comme c'est plus léger, on peut aussi les réparer plus facilement

Bad smell

"If it stinks, change it"

Exemples de ré-usinage #2

- **Introduce Explaining Variable:**
  - Expression complexe à comprendre en lisant la base de code
  - Assigner cette expression dans une variable qui explicite la sémantique.
- **Move method:**
  - Une méthode qui est plus utilisé par une classe que par celle où elle est définie
  - Déplacer la méthode pour augmenter la cohésion

Catalogue de mauvaises odeurs (Sonar)

- **Duplicated code**
  - If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them
- **Long method**
  - The key here is not method length but the semantic distance between what the method does and how it does it
- **Large Class**
  - When a class is trying to do too much, it often shows up as too many instance variables

Catalogue de mauvaises odeurs (Sonar)

- **Long Parameter list**
  - With objects you don't pass in everything the method needs; instead you pass enough so that the method can get to everything it needs
- **Divergent change**
  - One class is commonly changed in different ways for different reasons
- **Shotgun surgery**
  - Every time you make a kind of change, you have to make a lot of little changes to a lot of different classes

Catalogue de mauvaises odeurs (Sonar)

- **Feature envy**
  - A classic smell is a method that seems more interested in a class other than the one it actually is in
- **Data Clumps**
  - Bunches of data that hang around together really ought to be made into their own object
- **Primitive Obsession**
  - You can easily write little classes that are indistinguishable from the built-in types of the language

Catalogue de mauvaises odeurs (Sonar)

- **Switch statements**
  - The problem with switch statements is essentially that of duplication. Often you find the same switch statement scattered about a program in different places
- **Parallel Inheritance Hierarchies**
  - every time you make a subclass of one class, you also have to make a subclass of another
- **Speculative generality**
  - Oh, I think we need the ability to this kind of thing someday

Catalogue de mauvaises odeurs (Sonar)

- **Lazy Class**
  - A class that isn't doing enough to pay for itself should be eliminated. Often this might be a class that used to pay its way but has been downsized with refactoring. Or it might be a class that was added because of changes that were planned but not made. Either way, you let the class die with dignity
- **Temporary fields**
  - an object in which an instance variable is set only in certain circumstances

Catalogue de mauvaises odeurs (Sonar)

- **Message Chains**
  - You see message chains when a client asks one object for another object, which the client then asks for yet another object, which the client then asks for yet another another object, and so on.
- **Middle man**
  - You look at a class's interface and find half the methods are delegating to this other class
- **Alternative class with ≠ interfaces**
  - Any methods that do the same thing but have different signatures for what they do

Catalogue de mauvaises odeurs (Sonar)

- **Incomplete library class**
  - The trouble is that it is often bad form, and usually impossible, to modify a library class to do something you'd like it to do
- **Data class**
  - These are classes that have fields, getting and setting methods for the fields, and nothing else.
- **Refused Bedquest**
  - Subclasses get to inherit the methods and data of their parents. But what if they don't want or need what they are given? They are given all these great gifts and pick just a few to play with.

Les problèmes avec le ré-usinage

- Consomme du temps sur le temps de développement
- Peut avoir un impact sur les performances (a vérifier avec un banc de test)
- Est facilité lorsque l'outillage est adapté
  - Les IDE classiques proposent des outils de ré-usinage
  - Apprenez à les utiliser, c'est parfois "surprenant"
- Lien avec les architecture en couches :
  - On peut ré-usiner comme on veut tant qu'on ne touche pas aux interfaces publiques.