

**Introduction aux anti-patrons  
et au ré-usinage**

Sébastien Mosser - INF5153  
Chapitre 8 - Capsule 4  
Automne 2020

UQÀM | Département d'informatique

ace

credits photos: Pixabay

**Écrire du code  
de qualité,  
c'est compliqué**

UQÀM | Département d'informatique

MONKEYUSER.COM

**L'égo du développeur**

UQÀM | Département d'informatique



# Qu'est-ce qu'un anti-patron ?

---

---

---

---

---

---

---

---

## Identifier des défauts de conception



Un **BLOB** (ou **classe Dieu**) tend à **centraliser toute l'intelligence** du système, à **tout faire** et à **utiliser** des données en provenance de **classes purement structurelles**



---

---

---

---

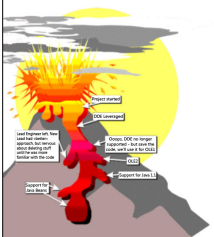
---

---

---

---

## Le fleuve de lave



- **Mauvaises décisions** prises lors du développement
- **"Code mort"** dans le produit
- Impossible de nettoyer, **les défauts se sont "solidifiés"**

---

---

---

---

---

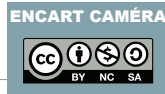
---

---

---

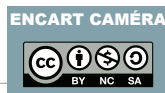


## Et tous les autres ...



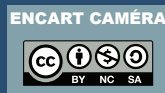
- Le **code spaghetti** / la **programmation copier-coller**
  - *Fort couplage, tout est entrelacé, cauchemar de maintenance*
- Le **Marteau doré** (Golden Hammer)
  - Un outil / une technologie / une méthodologie pour les gouverner tous.
    - Aussi appelé : "MVC everywhere", "RoR everywhere", ...
- La **balle d'argent** (Silver Bullet)
  - Recherche de la **solution ultime qui résoudra TOUS les problèmes**
    - *Même ceux qu'on connaît pas encore*

## Certains anti-patrons sont organisationnels



- **Bikeshedding** (popularisé par BSD)
  - **Discuter** longuement de **détails inutiles** (*le garage à vélo*)
  - Ne pas avoir le temps de **parler de ce qui est critique** (*la centrale nucléaire*)
  - Identifiée par Parkinson comme **Loi de la trivialité** (1957)
- Développement dirigé par les **signaux d'alarme**
  - **Attendre la catastrophe** pour faire plier les exigences de qualité
  - *"Si vous attendez la dernière minute, ça ne prend qu'une minute à faire"*

# 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

ENCART CAMÉRA



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

## Ré-usinage & Conception

ENCART CAMÉRA



- Le **ré-usinage** 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 **ré-usinage permet de réparer** ce qui n'as pas été vu lors de la conception
  - *Mais entre temps on a livré de la valeur*
- **Principe de l'architecture émergente**
  - *Requiert de l'expérience, et n'est pas adapté à toutes les équipes.*

## Pourquoi ré-usiner du code ?

ENCART CAMÉRA



- Les **programmes** ...
  - ... **complexes à lire** ... ;
  - ... avec de la **duplication** dans la logique d'affaire ...;
  - ... avec une **logique d'affaire pleine de conditions** ...
- ... **sont complexes 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)

# Importance des tests

ENCART CAMÉRA



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

## Règle de trois du ré-usinage

ENCART CAMÉRA



**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**

## Concrètement, c'est quoi un ré-usinage ?

ENCART CAMÉRA



- Peut **agir sur le code**, à plusieurs niveaux
  - 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 réusinage

ENCART CAMÉRA



- **Extract method :**
  - Fragment de code qui peut être sémantiquement regroupé avec d'autres
  - 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 de la méthode par le corps de celle-ci

## Les problèmes avec le ré-usinage

ENCART CAMÉRA



- **Consomme du temps** sur le temps de développement
- Peut avoir un **impact** sur les performances
  - (à 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.**

## Ré-usinage automatique

ENCART CAMÉRA



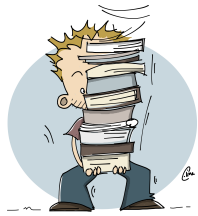
- On peut faire du **ré-usinage à la main**
  - **Cela demande de l'expertise et du temps**
- On peut aussi **automatiser ces opérations**
  - "Pourquoi écrire des programmes quand on peut écrire des programmes qui les écrivent à notre place ?"
- Exemples :
  - **Académique** : Repairator, un bot GitHub qui fixe des bugs et fabrique les PR
  - **Industriel** : Coccinelle, outil de réparation du code source de Linux

UQÀM

Département d'informatique

FACULTÉ DES SCIENCES  
Université du Québec à Montréal

ENCART CAMÉRA



<https://mosser.github.io/>



<https://ace-design.github.io/>

**Abonne toi à la chaine,**   
**et met un pouce bleu !** 

---

---

---

---

---

---

---

---