INF2050 – Outils et pratiques de développement logiciel

Refactoring

Jacques Berger

Objectifs

Introduire une pratique d‘amélioration du code

Prérequis

Programmation

Refactoring

Réécrire
Retravailler
Refaire le design
Refaire l'architecture

Modifier du code existant, sans modifier la fonctionnalité

Pourquoi?

Rendre le code plus lisible, plus clair

Faciliter le changement

Entretenir le système

Pourquoi?

Le code tend à se détériorer avec le temps

Plus on ajoute de la fonctionnalité, plus le code devient difficile à lire et à comprendre

Pourquoi?

Faire du refactoring dans un programme, c'est comme enlever les mauvaises herbes dans un jardin

Il faut le faire tôt, il faut le faire souvent, continuellement

Raffinement successif

Les méthodes Agile font la promotion du raffinement successif :

Faire du refactoring continuellement durant le développement du système

Si on doit travailler dans une classe, on la nettoie un peu avant pour faciliter notre travail

Quand?

Le code est mal placé :
    Variables, méthodes, classes

Les objets sont mal nommés

Duplication de code
    Principe DRY : Don't Repeat Yourself

Quand?

Design non orthogonal

- Un changement dans une classe entraîne un changement dans une autre classe

Connaissances dépassées

- On connait mieux le problème aujourd'hui

Quand?

Performance

- Déplacer le code pour améliorer les performances

Trop de responsabilités

- SRP : Single Responsibility Principle

Quand?

Conclusion

Tout ce qui semble mal peut être amélioré

Problèmes

La peur de toucher à ce qui fonctionne déjà

La réaction des patrons
«Ce code fonctionne mais je dois le réécrire»

Problèmes

Tendance : on fait le refactoring à la fin du projet

Réalité : on manque de temps, on coupe le refactoring

Problèmes

Dette technique

Comment?

Activité qu'il faut réaliser tranquillement, de façon délibérée et avec soin

On ne veut pas introduire de nouveaux bogues

Ne pas faire de refactoring et ajouter de la fonctionnalité en même temps

Comment?

Découper le refactoring en plusieurs petites étapes distinctes

- Déplacer un champ
- Fusionner 2 méthodes
- Déplacer une méthode
- Etc.

Faire autant de petites étapes que possible et tester après chaque petite modification

Tests

Toute manipulation de code peut introduire des erreurs dans le logiciel

- Le refactoring ne fait pas exception

La présence de tests unitaires favorise le refactoring

Tests

Les tests unitaires permettent de vérifier très rapidement si un refactoring a brisé la fonctionnalité en place

Ils favorisent le débogage et donnent confiance au développeur

Tests

Il est recommandé d'avoir une bonne couverture de tests avant de faire du refactoring

IDE

Les IDE modernes fournissent des fonctionnalités de refactoring automatiques

- Renommer des objets, des variables, etc.

Plus loin...

Refactoring – Improving the Design of Existing Code
Martin Fowler
Addison-Wesley Professional, 1999