La pratique du Refactoring

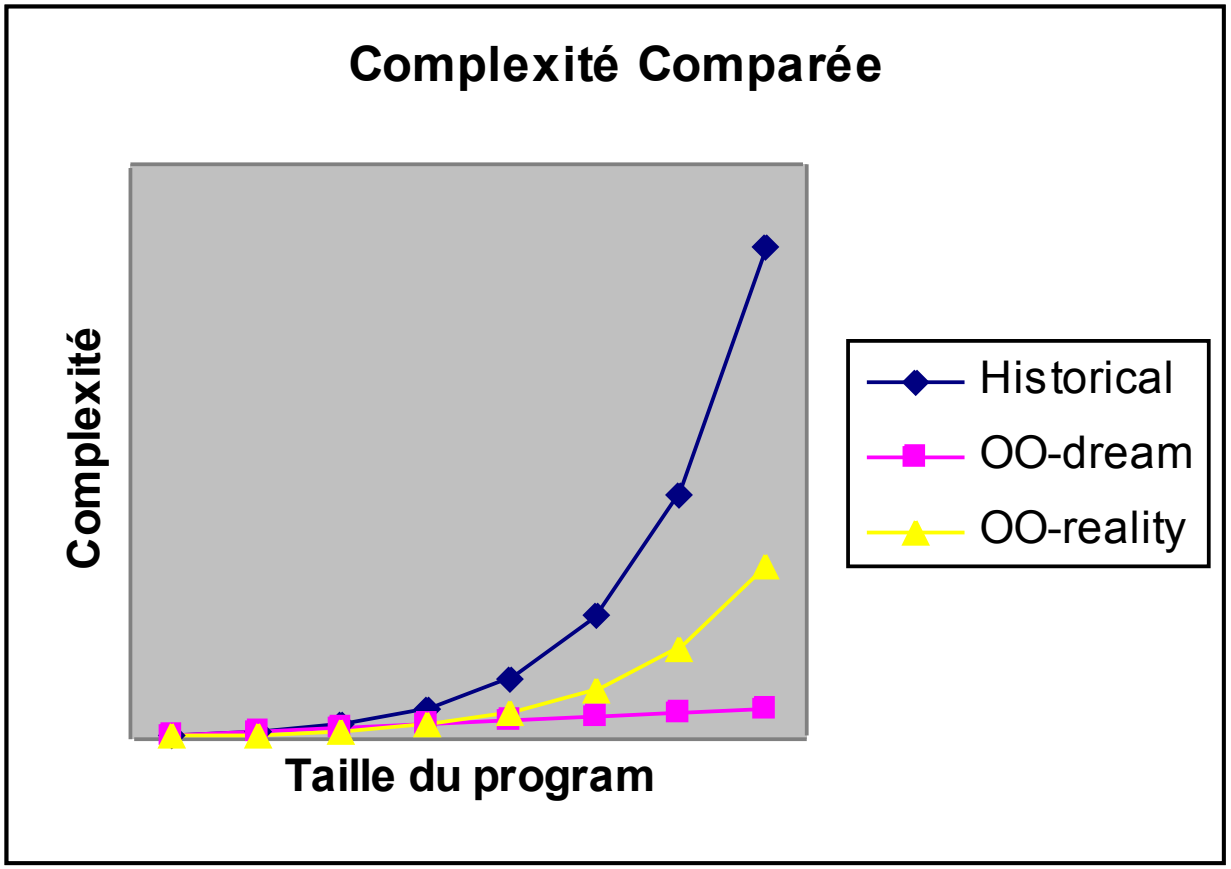
XP Day – 23 et 24 mars 2006

Hugh Prior
23 mars 2006

Sommaire

La cause

Le problème: l'inévitable et continuelle détérioration d'un logiciel

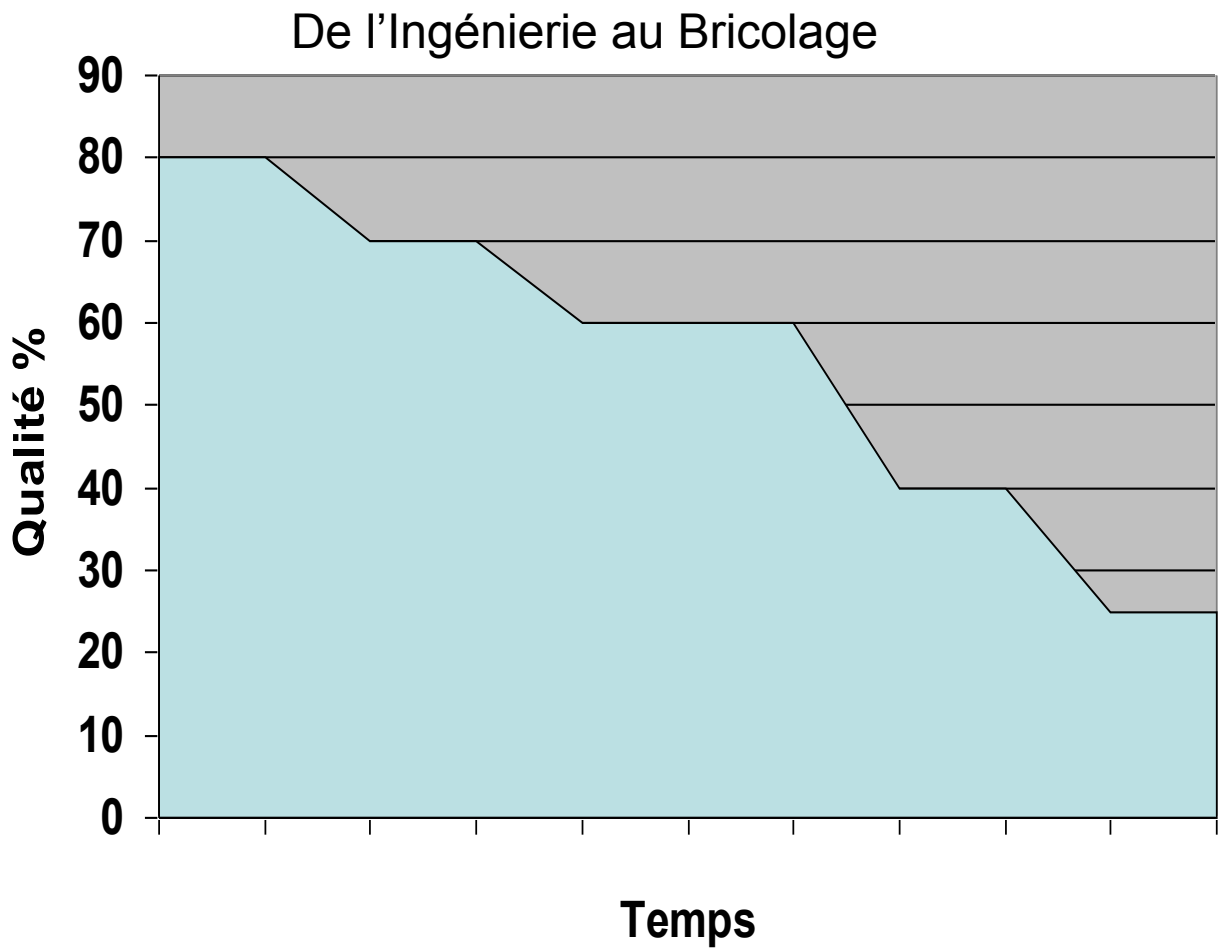


Les causes du problème

Les causes:

- Nouvelles fonctionnalités ajoutées tout le temps
- Changements de fonctionnalité
- Spécifications mauvaises ou incomplètes
- Changements dans l'équipe de développement
- Manque de connaissance du système
- Manque d'expérience des technologies
- Manque de temps

Le graphique de Qualité



Les effets

Les effets:

Code mal structuré

Code difficile à changer

Difficile à ajouter de nouvelles fonctionalités

Maintenir le code prend plus de temps

Les nouveaux développements prennent plus de temps

→ On attend toujours le "Nouveau" projet réécrit en partant de zéro

Faire face au problème

Ces facteurs négatif:

- **Ont un effet sur TOUT le développement de logiciels**
- **Ont un effet sur votre projet ACTUEL**
- **Vont avoir un effet sur votre PROCHAIN projet**

Le nier ne va pas vous aider!

Espérer simplement que tout se passe au mieux ne va pas vous aider!

Mais il y a une solution…

Chapitre: La Solution

“On ne se noie pas quand on tombe dans l’eau; on se noie quand on y reste."

— Edwin Louis Cole

La pièce des dossiers

Imaginez:

- **Une pièce énorme pleine de dossiers**
- **En désordre total**
- **Chaque dossier en désordre**
- **Une recherche prend toujours énormement de temps**

Définition du Refactoring

Définition du refactoring:

"Le processus consistant à modifier un système de telle façon que ça ne change pas le comportement externe du code mais que ça en améliore la structure interne"

***Refactoring: Improving the Design of Existing Code*, Martin Fowler**

Ce que c’est, ce que ce n’est pas

Ce que ce n’est pas:

- **Ajouter de la fonctionnalité**
- **Modifier de la fonctionnalité**
- **Améliorer la performance**

Ce que c’est:

- **Laisse le comportement externe identique à ce qu'il était avant les changements**
- **Change du code, que soit un petit ou un grand changement, mais avec le seul but d’améliorer la qualité interne du code**
- **Apporte un bénéfice essentiel mais pas visible: la productivité du programmeur**
- **S'effectue à chaque moment d’un projet**

Exemples

Exemples de Refactorings

- Renommer un variable, fonction ou un objet
- Extraction d'une méthode
- Réorganisation de la structure du code
- Ajouter ou enlever un paramètre
- Rendre les variables privées
- Créér une classe mère
- Convertir du code procedural en objet

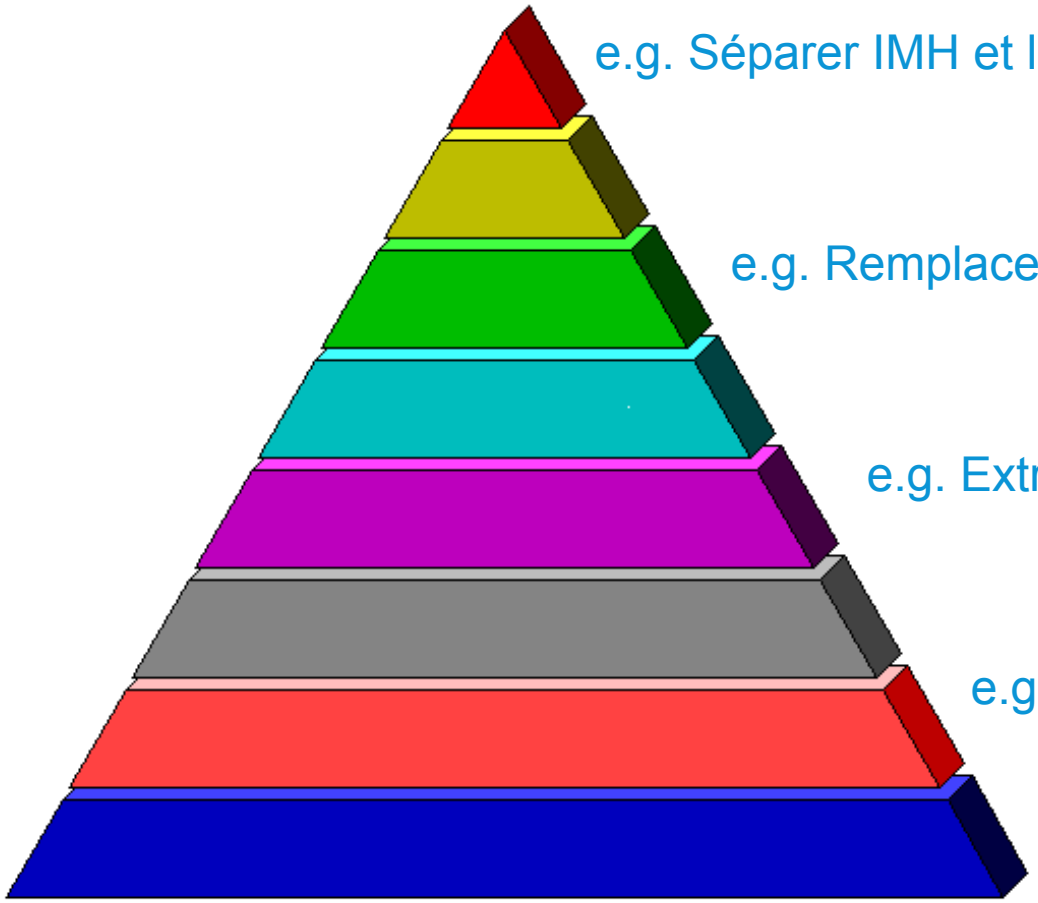
L'important en est le PRINCIPE plutôt que des exemples specifiques

Bonnes Nouvelles!

Les Bonnes Nouvelles!

- Le refactoring est quelque chose qu'on fait déjà parfois
- Le refactoring ne consiste pas à écrire du code complexe ou plein d'astuces, mais à écrire le meilleur code que nos compétences permettent
- 90% du temps sur 10% du code
- Chaque objet peut être refactoré independamment

Pyramide des refactorings



e.g. Séparer IMH et logique métier

e.g. Remplacer l'Héritage par la Délégation

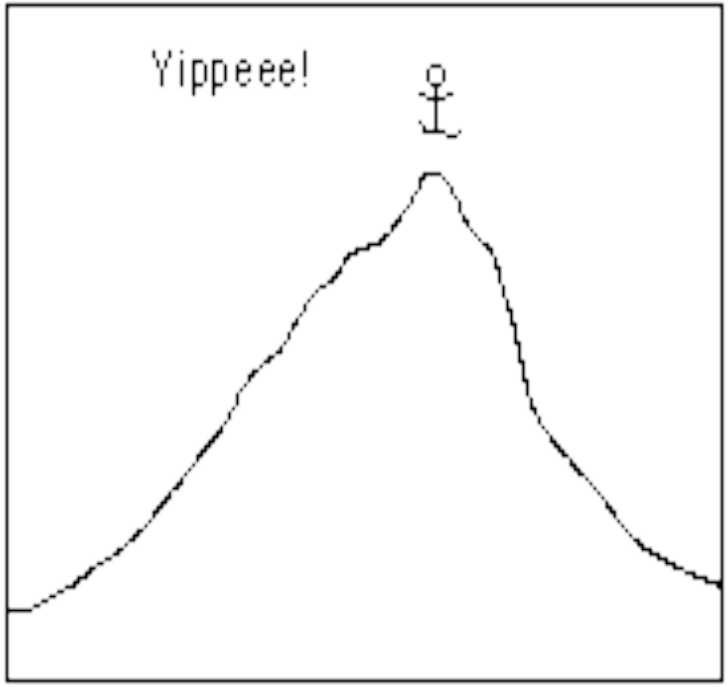
e.g. Extraire une classe mère

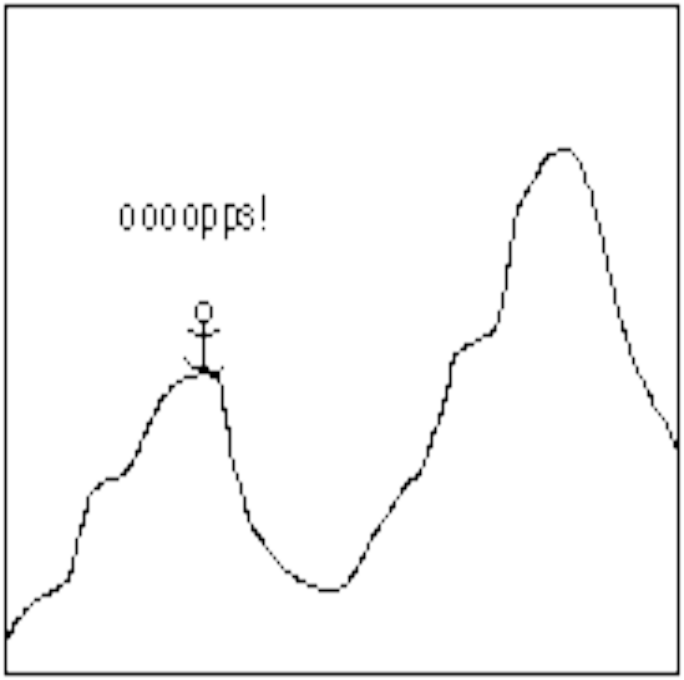
e.g. Extraire Méthode

e.g. Renommer une Variable

Atteindre le prochain sommet

Derrière beaucoup de refactorings d'autres sont cachés





On doit essayer de viser le sommet

Le refactoring – comment faire

1. Petits étapes
2. Un changement à la fois
3. Tester après chaque changement
4. Livrer fréquemment aux utilisateurs

Les Tests

Les Tests

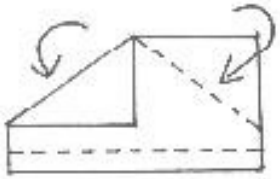
- **Petit changement, test, petit changement, test…**
- **Créér les meilleurs tests possible**
  - Ecrire auto-tests (test-driven development)
- **Accepter que les tests ne seront jamais aussi bien qu'on le voudrait**
- **Accepter que parfois le refactoring va génerer des bogues**
- **Utiliser le refactoring quel que soit le niveau d'utilisation des tests**
  - Le refactoring aide à reduire le nombre des bogues
  - Quand on a peur du refactoring, le logiciel est condamné

Refactoring du chapeau

1

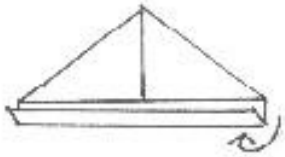
1. Plier le papier en deux, de sorte que vous avez un morceau de papier avec le bord long plié en haut.

2



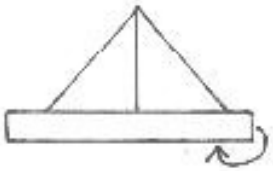
1. Pliez dans les coins comme dans le dessin pour que les bords arrivent au milieu pour former deux triangles

3



1. Pliez le bord en bas vers le haut pour qu'il vienne toucher les bords des deux triangles.

4



1. Pliez encore le bord en bas, avec le pli en bas des deux triangles.

Où faire des refactorings

Les “mauvais odeurs” dans le code

- **Code dupliqué**
- **Méthodes longues**
- **Saupoudrage**
- **Intimité impropre**
- **Les commentaires**
- **…**

Quand faire des refactorings

Quand faire des refactorings…

- **Opportunément**
- **Continuellement**
- **Systématiquement**
- **Audacieusement**
- **Courageusement**
- **Aggressivement**
- **Impitoyablement**

Quand ne pas faire de refactoring

La veille de la livraison

Si vous êtes sûr que personne ne touchera jamais au code après vous !

Conclusions

“Eviter le danger n’est pas plus sûr, à long terme, que s’exposer totalement.
La vie est soit une aventure audacieuse, ou rien."

— Helen Keller

Si ce n'est pas cassé...?

Historiquement: Des bonnes raisons pour se dire "Si ce n'est pas cassé... ne le répare pas"

Changement de paradigme

Aujourd'hui

- Langages OO
- Connaissance des programmeurs OO
- La méthodologie du refactoring
- Les outils pour aider au refactoring
- De meilleurs outils de test

Donc, on diramaintenant: "Si on peut l'écrire mieux, on fait du refactoring"

Objections

- Ca casse le code qui fonctionne
- La direction ne supporte pas le refactoring
- La peur
- Ne vaut pas le temps ni l'effort
- On n'a pas le temps pour le faire
- Nous n'avons pas de tests
- Manque de connaissance du refactoring (de la part des programmeurs ou des managers)
- La paressse

Synthèse

Le refactoring continuel est essentiel pour la santé à long terme d'un logiciel

Les points clés:

1. Petits étapes
2. Un changement à la fois
3. Tester après chaque changement
4. Livrer fréquemment

Refactorer: Opportunément, Continuellement, Systématiquement, Audacieusement, Courageusement, Aggressivement, Impitoyablement