LOG8371 : Ingénierie de la qualité en logiciel

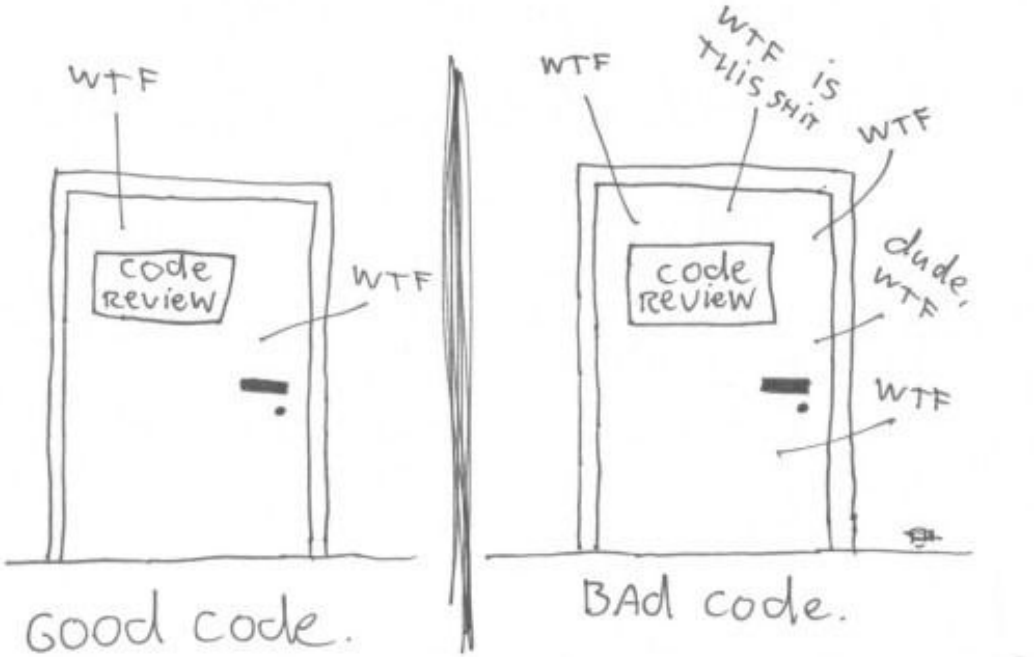
Qualité du Code - Code propre

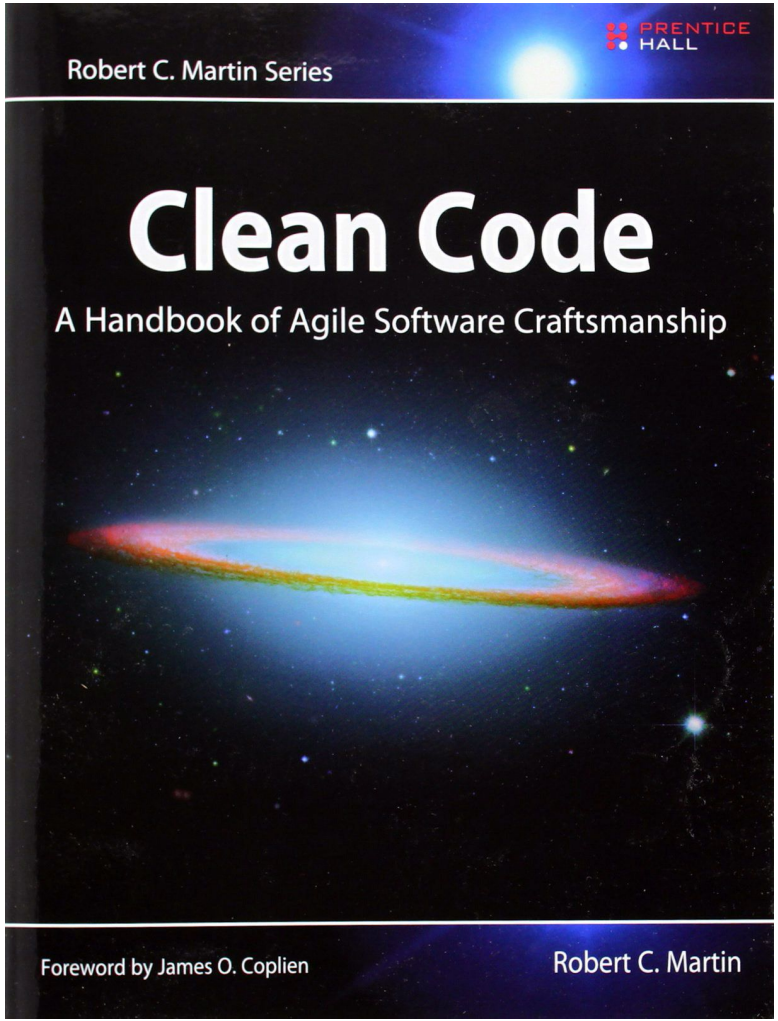
Hiver 2017

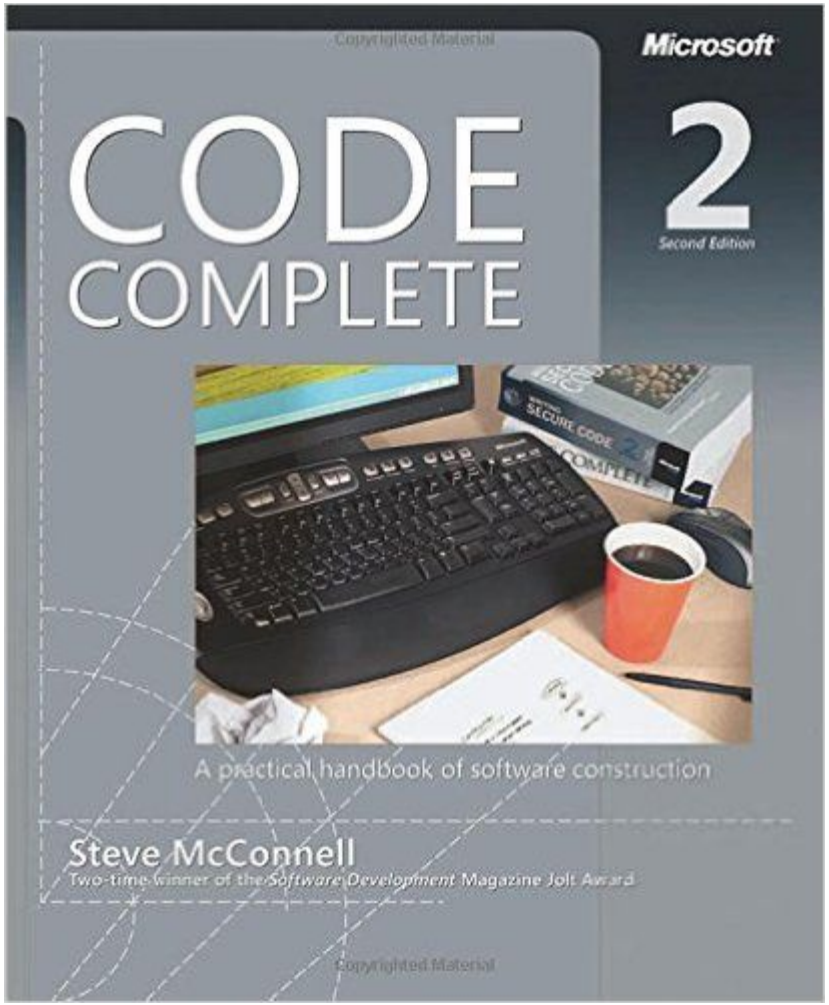
Fabio Petrillo
Chargé de Cours

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

The ONLY VALid MEASUREMENT OF Code QUALITY: WTFs/MINUTE




PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin


Microsoft
CODE COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell
Two-time winner of the Software Development Magazine Jolt Award

Code propre

- L'écriture d'un code propre nécessite l'utilisation **disciplinée** d'une myriade de **petites techniques** appliquées grâce au expertise de sens acquis de “**propreté**”.
- **Stratégie** pour appliquer ces techniques pour **transformer le code incorrect** en **code propre**, en appliquant une **série de transformations** jusqu'à ce qu'il devienne un système **élégamment** codé.
- Le code propre est **axé**, simple et direct.
- Le code propre semble toujours être écrit par **quelqu'un qui s'en occupe**.
- Diminution des **duplications**, grande **expressivité,** et la construction **rapide** des **abstractions simples**.

Noms significatifs

- Utilisez des noms révélateurs d'intention (et utilisez CamelCase)
  - int ~~d~~; // elapsed time in days
  - int **elapsedTimeInDays**;
- Éviter la désinformation
  - eg.: use *account**List*** only it is a List.
- Faire des distinctions significatives
  - public static void copyChars(char ~~a1~~[], char ~~a2~~[]) {
  - public static void copyChars(char **source**[], char **destination**[]) {
- Utiliser les noms prononcés
  - class ~~DtaRcrd102~~ { private final String ~~pszqint~~ = "102";
  - class **Customer** { private final String **recordId** = "102";
- Utiliser les noms recherchés
  - int **realDaysPerIdealDay** = 4;

Noms significatifs

- Évitez les encodages (éviter la notation hongroise ou les préfixes des membres)
  - Boolean ~~b~~Busy;
  - Boolean **busy**;
  - String ~~m_desc~~;
  - String **description**;
- Choisissez un mot par concept -> soyez cohérent!
  - fetch, retrieve or get? controller or manager?
- Évitez d'utiliser le même mot dans deux usage
- Utiliser les noms de domaine de la solution
  - Account**Visitor** -> visitor pattern!

Les fonctions

- Les fonctions devraient être **très petites**

  “Quelle est la meilleur taille (petite) d'une fonction? En 1999, je suis allé visiter Kent Beck à son domicile a l'Oregon. Nous nous avons fait de la programmation ensemble. À un moment, il m'a montré un joli petit programme Java/Swing qu'il appelait Sparkle. .... Lorsque Kent m'a montré le code, j'ai été choqué ***comment toutes les fonctions était petite****. J'avais l'habitude de faire des fonctions en Swing qui prennent des miles d'espaces verticaux.* ***Toutes les fonctions de son programme ne comportaient que deux, trois ou quatre lignes****. Chacun était clairement évident. Chacun a raconte une histoire. Et chacun vous conduit à la prochaine dans un ordre impérieux.* ***Voilà comment vos fonctions devraient être courtes!****”*

Les fonctions

- Les fonctions ne devraient faire qu'**une seule** et **propre** chose.
- Lire le code de **haut en bas**: la règle de Stepdown
- **Évitez** les déclarations de **commutation** -> utilisez les fonctionnalités abstraites et un polymorphisme.
- Les arguments de fonction -> plus de **deux arguments** doivent être évités.
- **Les flags d'arguments** sont **horrible**-> divisés en fonctions spécifiques sans arguments
- **N'a pas d'effets secondaires** -> promet de faire une chose, mais aussi d'autres choses **cachées**
- Les arguments de **sortie** devraient être **évités**
  - public void appendFooter(~~StringBuffer report~~)
  - report.appendFooter();
- **Préférer les exceptions** pour retourner les codes d'erreur

```
public void delete(Page page) {
  try {
    deletePageAndAllReferences(page);
  }
  catch (Exception e) {
    logError(e);
  }
}

private void deletePageAndAllReferences(Page page) throws Exception {
  deletePage(page);
  registry.deleteReference(page.name);
  configKeys.deleteKey(page.name.makeKey());
}

private void logError(Exception e) {
  logger.log(e.getMessage());
}
```

Mise en forme

- **Choisissez les règles de formatage et suivez-le!**
- **Classe de formatage vertical:** généralement 200 lignes de longueur; Limite supérieure de 500 lignes**.**
- **Chaque ligne** représente une expression ou une **clause,** et chaque **groupe de lignes** représente une **pensée complète.** Ces **pensées** devraient être **séparées** l'une de l'autre avec des **lignes vide.**
- **Gardez nos lignes courtes -> environ 45 caractères et plus que 80 (maximum 120).**
- **Utilisez l'indentation!**

Design - Raisons de créer une classe

- Modèle d'objets du **monde réel**
- Modèle d'objets abstraits
- **Réduire** et isoler la **complexité**
- **Masquer** les détails d'implémentation
- Effets limités des **changements**
- **Masquer** les données globales
- **Rationaliser** le passage des paramètres
- Créer des points de contrôle **centraux**
- Faciliter le code **réutilisable**
- Opérations liées aux **paquetages**

Qualité de classe - Liste de contrôle

- Types de données abstraites
  - Avez-vous pensé aux classes de votre programme en tant que types de données abstraites et évalué leurs interfaces à partir de ce point de vue?
- Abstraction
  - La classe a-t-elle un but central?
  - La classe est-elle bien nommée, et son nom décrit-t-il son but central?
  - L'interface de la classe présente-t-elle une abstraction cohérente?
  - L'interface de la classe montre-t-elle comment utiliser la classe?
  - Est-ce que l'interface de la classe est suffisamment abstraite pour que vous ne devez pas penser à la manière dont ses services sont mis en œuvre? Pouvez-vous traiter la classe comme une boîte noire?
  - Les services de la classe sont-ils suffisamment complets pour que d'autres classes ne se mêlent pas à leurs données internes?
  - Des informations non liées ont-elles été déplacées hors de la classe?

Qualité de classe - Liste de contrôle

- Abstraction (suite)
  - Avez-vous pensé à subdiviser la classe en composants de classes, et l’avez-vous subdivisé autant que vous le pouvez?
  - Êtes-vous en mesure de préserver l'intégrité de l'interface de la classe lorsque vous la modifiez?
- Encapsulation
  - Est-ce que la classe minimise l'accessibilité à ses membres?
  - La classe évite-t-elle d'exposer les données des membres?
  - Est-ce que la classe cache ses détails d'implémentation au autres classes autant que le langage de programmation le permet?
  - Est-ce que la classe évite de faire des hypothèses sur ses utilisateurs, y compris ses classes dérivées?
  - La classe est-elle indépendante d'autres classes? Est-ce faiblement couplé?

Qualité de classe - Liste de contrôle

- Héritage
  - L'héritage n'est-il utilisé que pour modéliser les relations “est un” - c'est-à-dire que les classes dérivées adhèrent-elles au principe de substitution de Liskov?
  - La documentation de la classe décrit la stratégie d'héritage?
  - Les classes dérivées évitent-elles “overriding” des routines “non-overridable”?
  - Les interfaces, les données et les comportements communs sont-ils aussi élevés que possible dans l'arbre d’héritage?
  - Les arbres d'héritage sont-ils assez profondes?
  - Tous les membres dans les classe basique sont-ils privés plutôt que protégés?
- Autres problèmes de mise en œuvre
  - Est-ce que la classe minimise les appels de routine indirectes vers d'autres classes?
  - Est-ce que toutes les données des membres sont initialisées dans le constructeur?
  - Avez-vous étudié les problèmes spécifiques à la langue pour les classes de ton language de programmation utilisee?

Routines de qualité - liste de contrôle

- La raison pour laquelle on crée la routine est-elle suffisante?
- Est-ce que toutes les parties de la routine qui profiteraient d'être mise dans leur routines ont été mises dans leurs routine?
- Le nom de la routine est-il solide, clair nom de verb-plus-objet pour une procédure ou une description de la valeur de retour d'une fonction?
- Le nom de la routine décrit-t-il tout ce que fait la routine?
- Avez-vous établi des conventions de dénomination pour les opérations courantes?
- La routine a-t-elle une cohésion solide et fonctionnelle - faire une seule chose qui est bien faite?
- Est ce que les routines ont perdu l'accouplement — Les connexions de la routine à d'autres routines sont-elles petites, intimes, visibles, et flexibles?
- La longueur de la routine est-elle déterminée naturellement par sa fonction et sa logique plutôt que par une norme de codage artificielle?

Routines de qualité - liste de contrôle des paramètres

- La liste de paramètres de la routine, prise dans son ensemble, présente-t-elle une abstraction d'interface cohérente?
- Les paramètres de la routine sont-ils dans un ordre raisonnable, y compris l'alignement de l'ordre des paramètres dans des routines similaires?
- La routine comporte-t-elle deux paramètres (max. Quatre)?
- Est-ce que chaque paramètre d'entrée est utilisé?
- Chaque paramètre de sortie est-il utilisé?
- La routine évite-t-elle d'utiliser les paramètres d'entrée comme variables de travail?
- Si la routine est une fonction, est-ce qu'elle renvoie une valeur valide dans toutes les circonstances possibles?

Outils de code propre

- FindBugs
  - http://findbugs.sourceforge.net/
- Checkstyle
  - http://checkstyle.sourceforge.net/
- PMD
  - http://pmd.sourceforge.net
- JBoss Tattletale
  - http://tattletale.jboss.org/
- UCDetector: Unnecessary Code Detector
  - http://www.ucdetector.org/