« Code smells »

D'après
Coder Proprement, Robert C. Martin, Pearson Education France

Bruno Mermet
2010

Présentation

- Définition de « code smell »

  « mauvaise odeur » émanant d'un code laissant présager des problèmes

- Voir aussi
  - Anti-patterns
  - Refactoring

Commentaires (1)

- C1 : informations inappropriées
  - Caractéristique : commentaire contenant des choses qui devraient être ailleurs (dans gestionnaire de version, etc.)
  - Remède : supprimer, et adopter un gestionnaire de version si ce n'est déjà fait
- C2 : commentaires obsolètes
  - Caractéristique : commentaire faisant référence à des choses qui n'existent plus
  - Remède : mettre à jour ou supprimer le commentaire
- C3 : commentaire redondant
  - Caractéristique : commentaire paraphrasant le code
  - Remède : supprimer (ou rajouter ce qui manque)

Commentaires (2)

- C4 : commentaires mal rédigés
  - Caractéristique : commentaire pas ou peu compréhensible
  - Remède : rédiger ses commentaires avec soin
- C5 : code mis en commentaire
  - Caractéristique : évident
  - Remède : supprimer !
- Remarque générale
  - Ecrire son code de manière à réduire au maximum la nécessité de commentaires
    - Bien placer les méthodes dans les classes et les classes dans les packages
    - Donner des noms clairs aux classes, méthodes et variables

Environnement

- E1 : la construction exige plusieurs étapes
  - Caractéristique : construire le programme doit se faire en plusieurs étapes, pas évidentes
  - Remède : adopter des outils de constructions adéquates (Maven)
- E2 : Les tests exigent plusieurs étapes
  - Caractéristique : évident
  - Remède : utiliser des outils comme Junit (avec TestSuite) et/ou maven

Fonctions/Méthodes (1)

- F1 : trop grand nombre d'arguments
  - Caractéristique : pas plus de 3 arguments. 0 ou 1, c'est bien ; 2, c'est à voir ; 3 ; c'est limite.
  - Remède : revoir le code ; éventuellement, introduire une nouvelle classe ou décomposer la méthode
- F2 : arguments de sortie
  - Caractéristique : un argument d'une méthode est modifié par la méthode pour renvoyer une valeur
  - Remède : mettre la méthode dans l'objet modifier ou renvoyer une valeur de retour ou décomposer la méthode

Fonctions/Méthodes (2)

- F3 : arguments indicateurs
  - Caractéristique : un argument (typiquement booléen) sert à modifier le comportement de la fonction
  - Remède : éclater la fonction en plusieurs différentes
- F4 : fonction morte
  - Caractéristique : une fonction n'est jamais appelée
  - Remède : supprimer la fonction

Généralités (1)

- G1 : multiples langages dans un même fichier source
  - Caractéristique : mélange de langages dans un fichier source (jsp/java, etc.)
  - Remède : simplifier ou mieux structurer son code
- G2 : comportement évident non implanté
  - Caractéristique : une fonction n'implante pas un comportement auquel on pourrait s'attendre logiquement vu son nom
  - Remède : compléter la fonction

Généralités (2)

- G3 : comportement incorrect aux limites
  - Caractéristique : le code fonctionne normalement dans la plupart des cas, mais incorrectement aux bornes des domaines de définition ou sur les valeurs clés dans le code
  - Remède : ne pas se limiter à ses intuitions ; prévoir des cas de test pour chaque cas aux limites
- G4 : neutralisation des sécurités
  - Caractéristique : des codes de sécurité ont été mis en commentaire ou sont évités ou des avertissements du compilateur sont ignorés
  - Remède : éviter tout cela

Généralités (3)

- G5 : redondance
  - Caractéristique : on trouve plusieurs fois la même chose dans le code
  - Remède : factoriser les traitements ; utiliser les Design Patterns adéquats
- G6 : code au mauvais niveau d'abstraction
  - Caractéristique : une méthode mélange des actions de haut et bas niveaux ou mauvaise répartition entre une classe abstraite et son implantation
  - Remède : mieux structurer le code, en regroupant par exemple les actions bas niveau dans une méthode intermédiaire ou re-positionnement des méthodes entre les classes abstraites et concrètes

Généralités (4)

- G7 : classes de base dépendant de classes dérivées
  - Caractéristique : Une classe fait explicitement référence (via un switch ou un if) à ses classes dérivées
  - Remède : utiliser le polymorphisme et la redéfinition
- G8 : beaucoup trop d'informations
  - Caractéristique : beaucoup de méthodes (ou de variables d'instance) dans une seule et même classe
  - Remède : mieux structurer le code ; éclater la classe en classes plus petites

Généralités (5)

- G9 : code mort (voir F4 : fonction morte)
  - Caractéristique : une partie du code derrière un test n'est jamais exécuté
  - Remède : si c'est normal, supprimer le code en question
- G10 : séparation verticale
  - Caractéristique : une variable/méthode est déclarée loin de sa première utilisation
  - Remède : réduire la taille des méthodes ou des fonctions

Généralités (5)

- G9 : code mort (voir F4 : fonction morte)
  - Caractéristique : une partie du code derrière un test n'est jamais exécuté
  - Remède : si c'est normal, supprimer le code en question
- G10 : séparation verticale
  - Caractéristique : une variable/méthode est déclarée loin de sa première utilisation
  - Remède : réduire la taille des méthodes ou des fonctions

Généralités (6)

- G11 : incohérence
  - Caractéristique : les méthodes/variables ne sont pas nommées de façon uniforme
  - Remède : mieux gérer ses règles de nommage
- G12 : désordre
  - Caractéristique : du code, des commentaires, des variables, ne servent à rien
  - Remède : nettoyer périodiquement le code

Généralités (7)

- G13 : couplage artificiel
  - Caractéristique : on hérite d'une classe juste pour tirer partie de ses constantes par exemple
  - Remède : fonctionner autrement (!)
- G14 : envie de fonctionnalité
  - Caractéristique : une méthode appel beaucoup d'accesseurs d'un objet (passé en paramètre ou variable d'instance)
  - Remède : transférer la méthode (ou une partie) dans la classe de l'objet en question

Généralités (8)

- G15 : argument sélecteur (voir F3 : argument indicateur)
- G16 : Intentions obscures
  - Caractéristique : une partie du code est incompréhensible
  - Remède : mieux détailler le code en le structurant mieux ou en décomposant les formules avec des variables aux noms explicites
- G17 : Responsabilité mal placée (voir G14)
  - Caractéristique : une méthode n'est pas dans la classe où on s'attend à la trouver, ou une classe dans le package attendu
  - Remède : déplacer la méthode ou la classe

Généralités (9)

- G18 : méthodes statiques inappropriées
  - Caractéristique : une méthode statique modifie/consulte un de ses paramètres (voir F2 : arguments de sortie)
  - Remède : transformer la méthode en méthode d'instance de la classe du paramètre modifié
- G19 : utiliser des variables explicatives (voir G16, C)
  - Caractéristique : une expression/un code est incompréhensible
  - Remède : décomposer les expressions, renommer les variables

Généralités (10)

- G20 : Les noms des fonctions doivent indiquer leur rôle
  - Caractéristique : un nom de fonction est ambigüe ou non conforme à son action
  - Remède : choisir un nom plus clair
- G21 : comprendre l'algorithme
  - Caractéristique : l'algorithme (pas sa justification) n'est pas bien compris
  - Remède : toujours attendre de comprendre un algorithme avant de l'implanter

Généralités (11)

- G22 : rendre physique les dépendances logiques
  - Caractéristique : un module ne fonctionne que par une hypothèse qu'il fait sur le fonctionnement d'un autre
  - Remède : transformer cette hypothèse en un appel de méthode pour obtenir l'information
- G23 : préférer le polymorphisme aux tests (voir G7)
- G24 : respecter des conventions standards (voir G11)
- G25 : éviter les nombres magiques
  - Caractéristique : le code fait figurer des nombres obscurs sans explication
  - Remède : utiliser des constantes à la place

Généralités (12)

- G26 : être précis
  - Caractéristique : utiliser toujours le bon type au bon endroit
  - Remède : limiter les types réels, choisir bien ses classes génériques
- G27 : privilégier la structure à une convention
  - Caractéristique : la garantie de bon fonctionnement dépend d'un respect d'un standard d'écriture
  - Remède : utiliser une structure qui garantit le bon fonctionnement

Généralités (13)

- G28 : encapsuler les expressions conditionnelles
  - Caractéristique : un test dépend de plusieurs conditions
  - Remède : résumer la signification de toutes ces condition par un nom de méthode effectuant le calcul de l'expression booléenne
- G29 : éviter les expressions conditionnelles négatives
  - Caractéristique : un test est de la forme if (!cond())
  - Remède : transformer cond en condf pour que le test devienne if (condf())

Généralités (13)

- G30 : les fonctions doivent faire une seule chose
  - Caractéristique : une fonction fait plusieurs choses à la fois
  - Remède : éclater la fonction en plusieurs
- G31 : couplages temporels cachés
  - Caractéristique : le bon fonctionnement d'une méthode dépend implicitement de l'appel préalable d'une autre méthode
  - Remède : Rendre le couplage explicite

Généralités (14)

- G32 : ne pas être arbitraire
  - Caractéristique : certaines structures du code imposent des choses sans raison
  - Remède : revoir la structure pour supprimer ce qui n'est pas nécessaire
- G33 : encapsuler les conditions aux limites
  - Caractéristique : le traitement d'une condition aux limites est disséminé dans le code
  - Remède : le regrouper dans une méthode, voire une classe (Pattern *Cas Particulier*)

Généralités (15)

- G34 : les fonctions doivent descendre d'un niveau d'abstraction (voir G6)
  - Caractéristique : différence de niveaux d'abstraction dans une méthode
  - Remède : introduire une ou plusieurs méthodes intermédiaires
- G35 : conserver les données configurables à un niveau élevé
- G36 : éviter la navigation transitive (loi de Déméter)
  - Caractéristique : un code contient a.getB().getC().m()
  - Remède : introduire la méthode nécessaire dans la classe de a.

Java

- J1 : éviter les longues listes d'importation et préférer le '*' (discutable)
- J2 : Ne pas hériter des constantes
- J3 : Privilégier les énumérations aux constantes

Noms

- N1 : choisir des noms descriptifs
- N2 : choisir des noms au niveau d'abstraction adéquat
- N3 : employer si possible une nomenclature standard
- N4 : utiliser des noms non ambigus
- N6 : éviter la codification
- N7 : décrire les effets secondaires dans les noms

Tests

- T1 : tests insuffisants
- T2 : utiliser un outil d'analyse de couverture
- T3 : ne pas omettre les tests triviaux
- T4 : un test ignoré est une interrogation sur une ambiguïté
- T5 : Tester aux limites
- T6 : tester de manière exhaustive autour des bugs
- T7 : les motifs d'échec sont révélateurs
- T8 : les motifs dans la couverture des tests sont révélateurs
- T9 : les tests doivent être rapides