RAPPORT FINAL PROJET IFT3051

DETECTION MANUELLE DES ANTI-PATRONS DE CONCEPTION DANS NUTCH VERSION 0.7.1

TRAVAIL ÉFFECTUÉ PAR :
TRAORÉ FATOUMATA
TRAF12558403

RAPPORT PRÉSENTÉ AU PROFESSEUR YANN-GAËL GUÉHÉNEUC

UNIVERSITE DE MONTREAL

HIVER2007

TABLE DES MATIÈRES

OBJECTIF : le but de ce travail est de détecter manuellement les anti-patrons de conception dans la version 0.7.1 de Nutch pour vérifier et valider les résultats obtenus par l'outil conçut par le professeur Yann-Gaël Guéhéneuc et Naouel Moha, étudiante au doctorat, et ce, par une comparaison entre les résultats obtenus par cet outil et ceux obtenus manuellement.

I-INTRODUCTION

Commençons par une brève présentation de Nutch, puis quelques définitions de base sur les anti-patrons.

I.1) PRESENTATION DE NUTCH

Nutch est un projet qui a pour objectif la conception d'un moteur de recherche à code source libre du genre Google mais en restant une entreprise a but non commercial pour contrecarrer les intérêts premiers des entreprises qui le sont.

Nutch survit donc grâce à la contribution des développeurs bénévoles qui veulent bien y participer ainsi qu'à des dons de particuliers.

I.2) LES ANTI-PATRONS

Les anti-patrons sont des problèmes courants de conceptions dûs soit au non-usage des ***patrons de conception***, soit au mauvais usage de ceux-ci, soit à une mauvaise conception. Les patrons de conceptions décrivent une bonne organisation de classes à suivre pour avoir un code plus efficace et plus utilisable.

La présence d'anti-patrons entraine généralement dans un programme une lenteur excessive dans le logiciel, des couts de réalisation ou de maintenance élevés, des comportements anormaux et la présence de bogues.

L'avancée de la technologie, l'informatisation des systèmes, le fort besoin d'innovation a entraîné une forte demande en logiciel. C'est exactement dans ce contexte que nous nous plaçons lorsque nous pensons

qu'il est essentiel pour chaque programmeur de concevoir un programme à la fois performant mais aussi utilisable et efficace. C'est dans ce souci de qualité du logiciel que nous étudions les anti-patrons afin de les éviter autant que possible.

II-DIFFERENTS TYPES D'ANTI-PATRONS

II-1. LES ANTI-PATRONS

II-1.1 le Blob

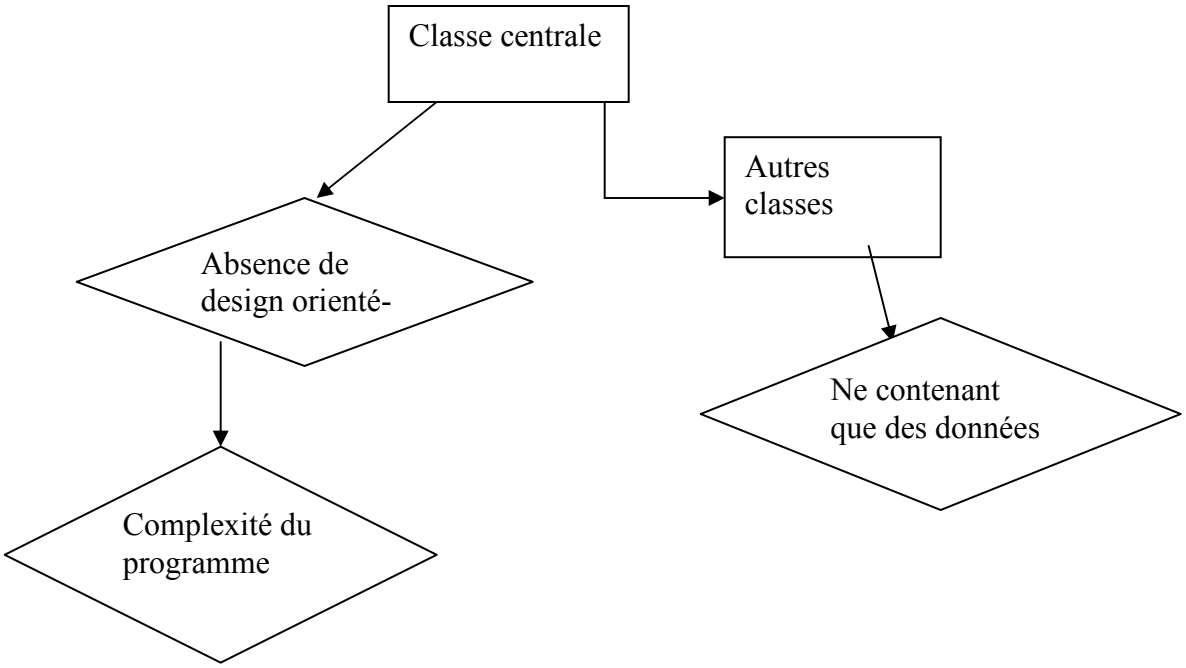
Ce type d'anti-patron est rencontré dans des conceptions où une classe centrale monopolise le processus tandis que les autres classes comportent juste quelques données. Dans ce cas-ci la classe centrale toutes les responsabilités. Elle se caractérise donc souvent par la présence d'une grosse classe.

Comment reconnaître le ' Blob '

Un blob est une classe qui :
-possède un trop grand nombre d'attribut ou/et d'opérations (>=60).
-définit une grande collection d'attributs et d'opérations sans lien (entrainant un manque de cohérence) définis dans une même classe.
-est liée à plusieurs classes ne contenant que des données.
-manque d'une certaine structure orientée-objet.
-entraîne souvent des modifications ailleurs lorsqu'elle est modifiée.
-est parfois très complexe.

Structure du Blob

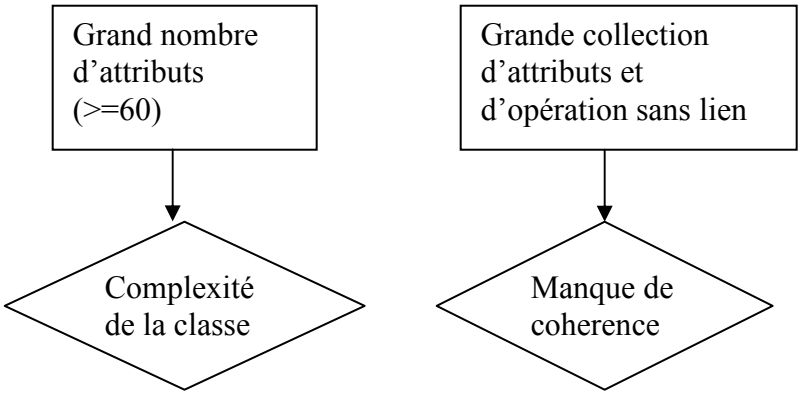
→ Niveau conception



→ Niveau flot d'exécution

S/O

→ Niveau des classes



→ Niveau méthode

S/O

II-1.2 le Lava Flow

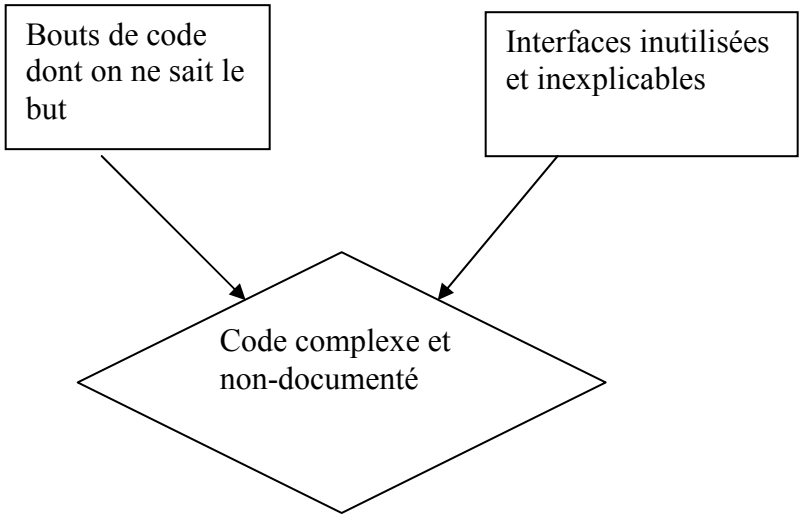
Ce type d'anti-patron est caractérisé par le fait que, dans un programme, aucun des développeurs concernés n'arrive à dire à quoi sert un certain bout du code. C'est à dire que ce code ne contient pas de commentaire et donc les programmeurs ne comprennent pas a quoi il sert et, de plus, ils pensent ne pas en avoir besoin sauf que le code a l'air tellement complexe qu'ils sont persuadés que sa suppression pourrait entrainer plus tard des erreurs.
Cela est généralement dû au fait que l'un des programmeurs a écris ce code en prévision d'une certaine situation avant même de l'avoir rencontrée.

Comment reconnaître le Lava Flow

Le Lava Flow est une classe qui :
-contient Plusieurs variables ou bouts de code dans le programme dont on ne connait le but.
-contient du code complexe et non-documenté, des classes et des fonctions ayant l'air importantes qui ne sont pas clairement liées à l'architecture du programme.
-possède des blocs entiers de codes sans explication ni documentation.
-a plusieurs fois dans le code des commentaires ' à remplacer '.
-possède du code non utilisé mais laissé dans le programme.
-contient des interfaces inutilisées et inexplicables.

Structure du Lava Flow

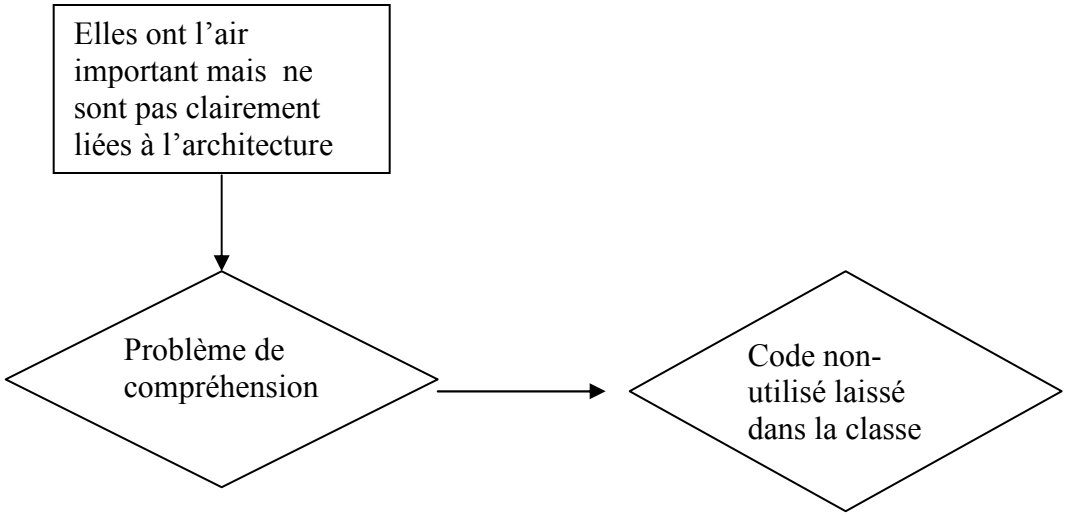
→ Niveau conception



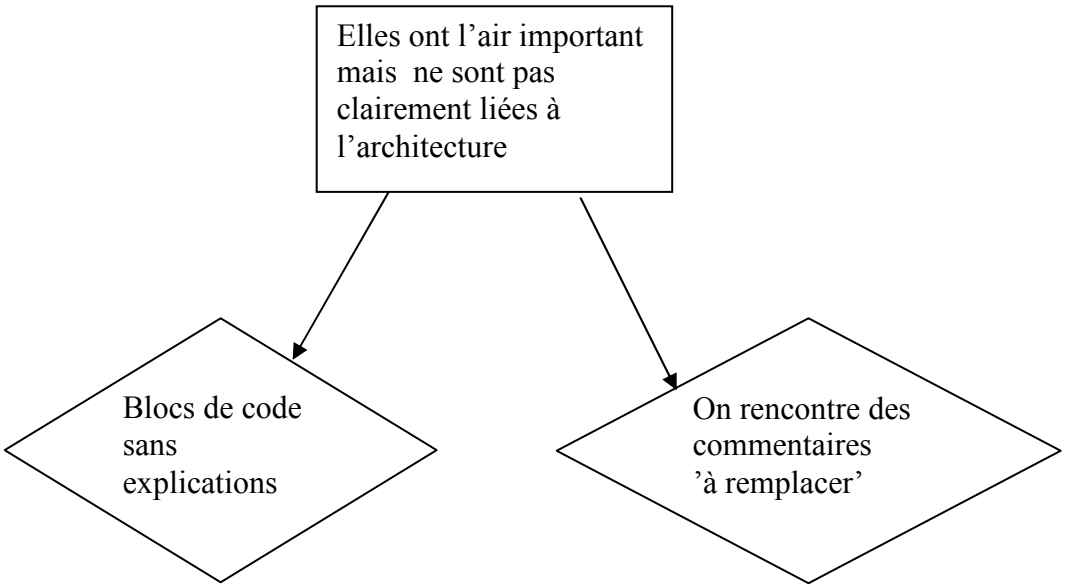
→ Niveau flot d'exécution

S/O

→ Niveau des classes



→ Niveau méthode



II-1.3 le Functional Decomposition

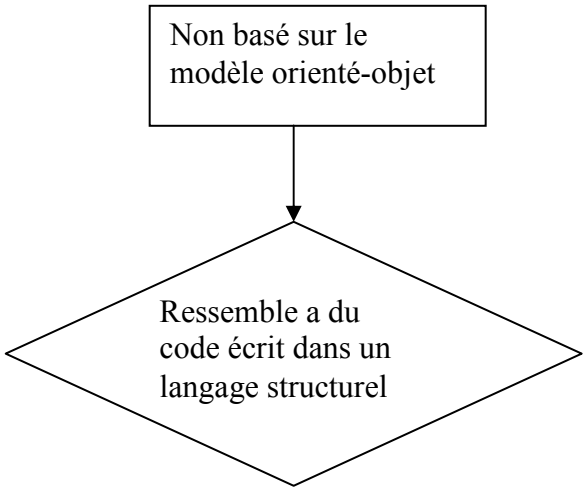
On rencontre généralement ce type d'anti-patron lorsqu'un développeur non orienté-objet programme dans un langage orienté-objet. Lorsque les développeurs sont à l'aise avec un programme principal (main) qui appelle de nombreux sous-programmes, ils essaient de faire de chaque sous-programme une classe, ignorant ainsi la conception orienté-objet. Le code résultant ressemble finalement à un code écrit dans un langage structurel.

Comment reconnaitre le Functional Decomposition

Le Functional Decomposition est une classe qui :
-a des noms de fonction du genre "Calculate_Interest" ou "Display_Table".
-possède que des attributs de classe 'private' qui ne sont utilisés qu'à l'intérieur de la classe où ils sont déclarés.
-contient des classes qui font seulement une action, par exemple avoir une seule fonction.
-a une architecture qui n'est définitivement pas basée sur le modèle orienté-objet (par exemple avec le polymorphisme ou l'héritage).

Structure du ' Functional décomposition '

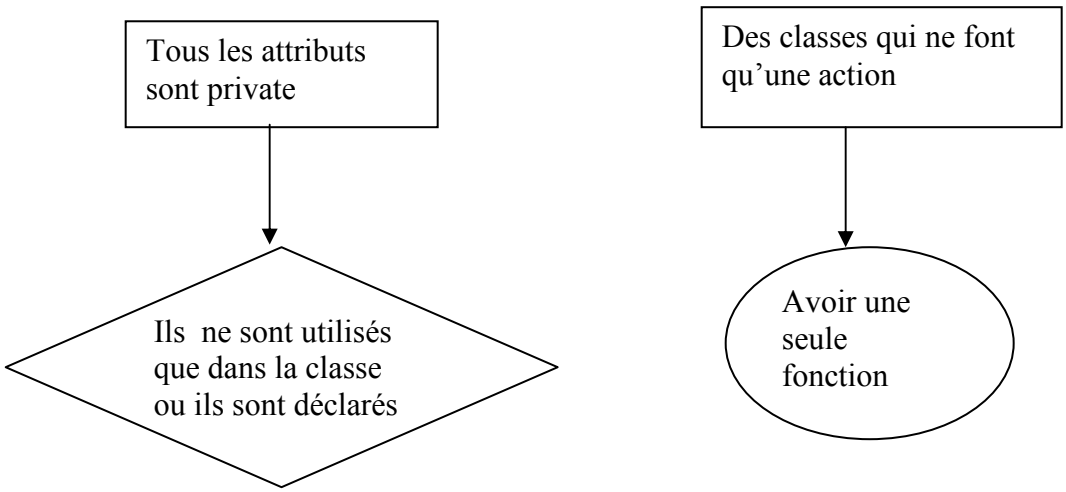
→ Niveau design



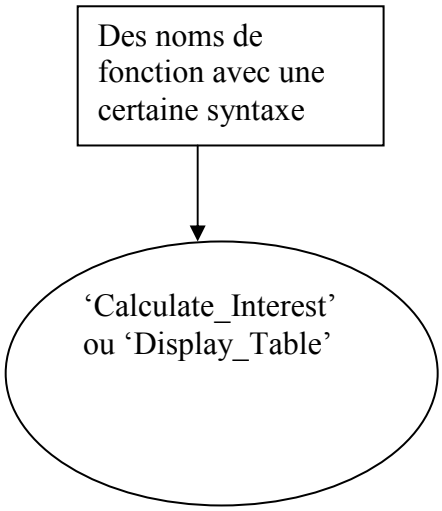
→ Niveau flot d'exécution

S/O

→ Niveau de La classe



→ Niveau de la méthode



II-1.4 le Poltergeist

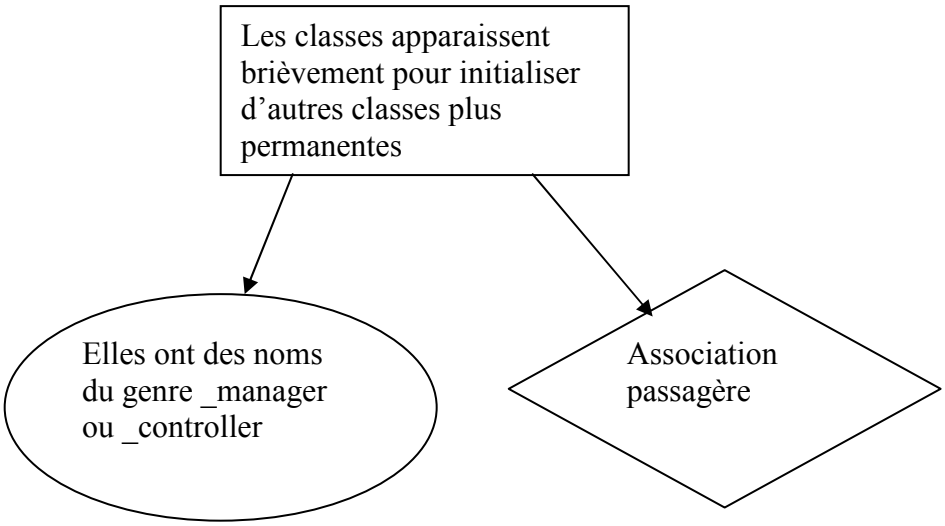
Le Poltergeist vient du fait que certains programmeurs utilisent l'effet secondaire de certains langages 'pour accomplir une tache principale est une mauvaise utilisation du langage et de l'architecture qui doit être évitée.
Cet anti-patron est typique lorsque des développeurs familiers avec les modèles de processus mais débutants avec l'orienté-objet définissent la conception.

Comment reconnaître le Poltergeist

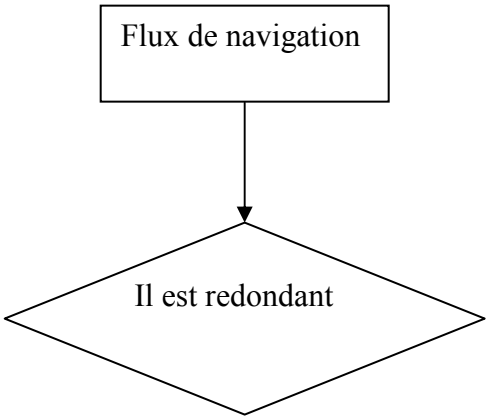
Le Poltergeist est une classe qui:
-possède des chemins de navigation complexes.
-possède des associations passagères.
-est sans états.

Structure du 'Poltergeist'

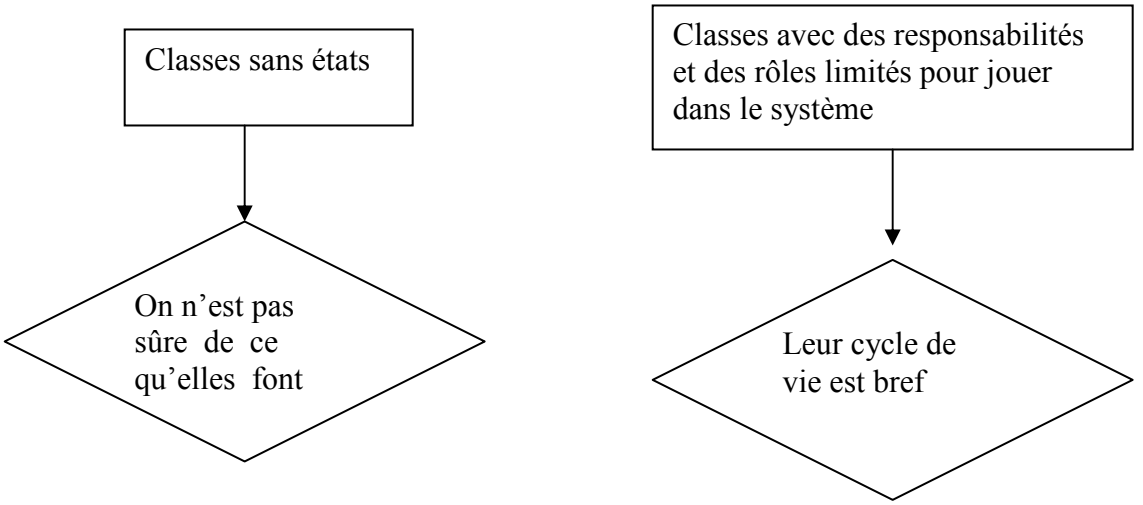
→ Niveau de la conception



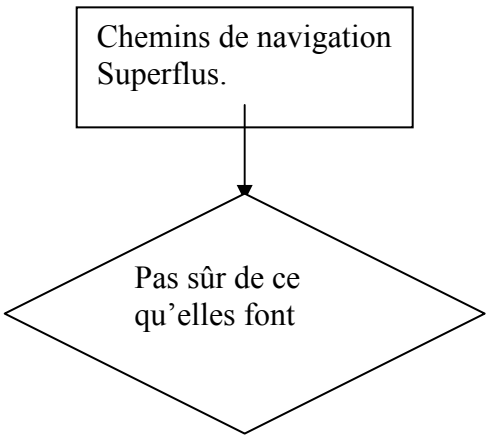
→ Niveau flot d'exécution



→ Niveau de la classe



→ Niveau de la méthode



II-1.5 Golden Hammer

Le Golden Hammer résulte de la mauvaise application d'un concept ou d'un outil préféré. C'est à dire que les développeurs sont à l'aise avec une certaine approche, ils ne sont donc pas disposés à en apprendre une autre qui serait peut être meilleure que celle adoptée.

Comment reconnaitre le Golden Hammer

Le Golden Hammer est dû au fait que :

-des produits et des outils identiques sont utilisés pour la sélection de divers produits.
-les solutions des outils ont de faibles performances et faibles adaptabilités ainsi que d'autres problèmes face aux autres solutions dans l'industrie.
-l'architecture du système est mieux décrite par un produit particulier, un progiciel ou par un fournisseur d'outil.

Structure du Golden Hammer

→ Niveau de la conception
S/O

→ Niveau flot d'exécution
S/O

→ Niveau de la classe
S/O

→ Niveau de la méthode
S/O

II-1.6 Le Spaghetti Code

Le Spaghetti Code est un anti-patron qui apparaît quand un programme ou un système ne contient qu'une petite structure logicielle. Le codage et les nombreux branchements d'une partie du code à une autre compromettent la structure du logiciel à tel point qu'elle manque de clarté, même pour le programmeur original, s'il s'éloigne du code pour une certaine durée.

Comment reconnaitre le Spaghetti Code

Le Spaghetti Code est une classe qui :

-possède des méthodes très orientée-processus en fait les objets sont nommées comme des processus.
-possède peu de relation entre les objets.
-a plusieurs méthodes objets qui ne possèdent pas de paramètres et qui sont donc obligées d'utiliser des classes ou des variables globales.
-est difficile à réutiliser et quand on le fait c'est souvent par copier/coller.

Structure du Spaghetti Code

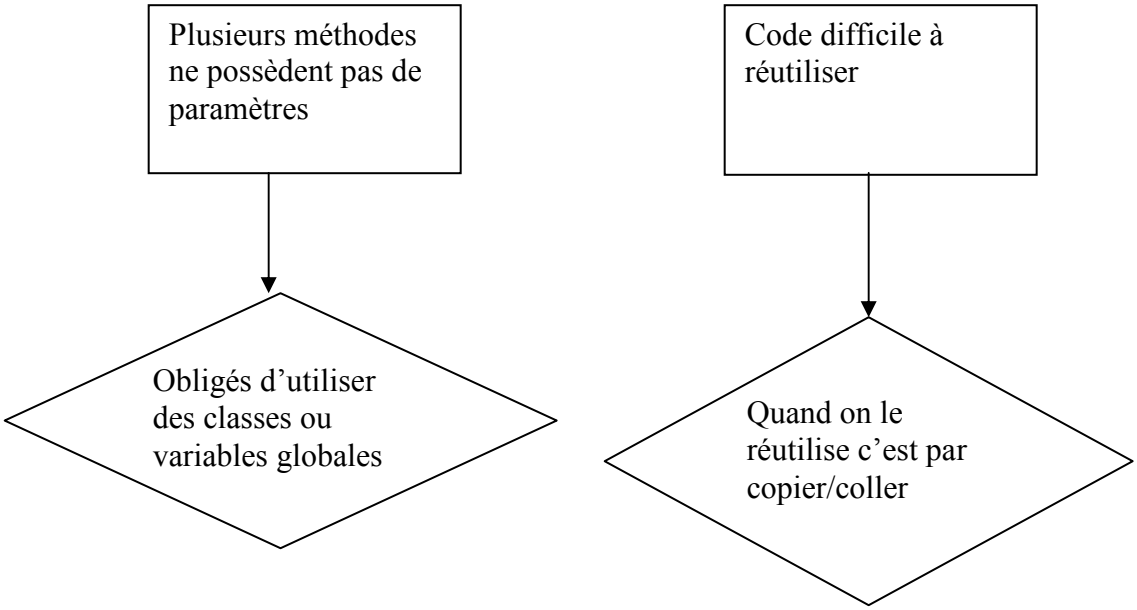
→ Niveau de la conception

S/O

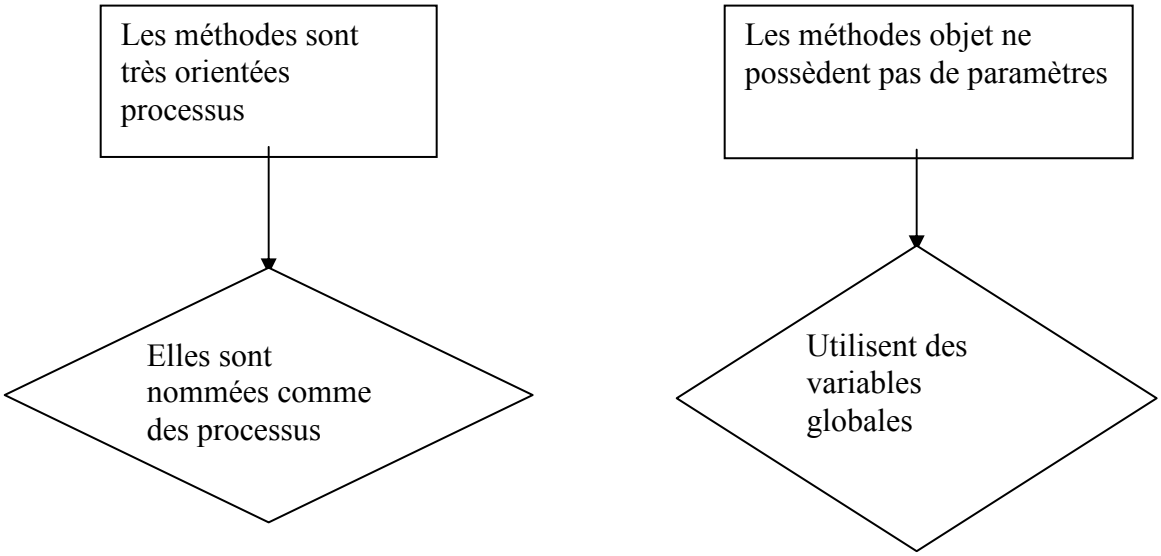
→ Niveau flot d'exécution

S/O

→ Niveau de la classe



→ Niveau de la méthode



II-1.7 Le Cut-and-Paste Programming

Le Cut-and-Paste Programming se caractérise par la présence de plusieurs bouts de code semblable dans un même projet. Cela est dû au fait que plusieurs programmeurs travaillent ensemble sur un même projet en suivant des exemples de développeurs plus expérimentés. Finalement les programmeurs modifient du code qui fonctionnait sur un cas similaire et donc ils l'adaptent pour supporter de nouveaux types de données ou de nouveaux comportements, ce qui entraine des duplications de code.

Comment reconnaître le Cut-and-Paste Programming

Le Cut-and-Paste Programming est une classe où:
-les mêmes erreurs se répètent dans le programme.
-le nombre de ligne de code augmente sans améliorer la productivité ni la performance du programme.
-il est difficile de localiser et de corriger toutes les instances d'une erreur donnée.
-le code est considéré comme auto documenté.

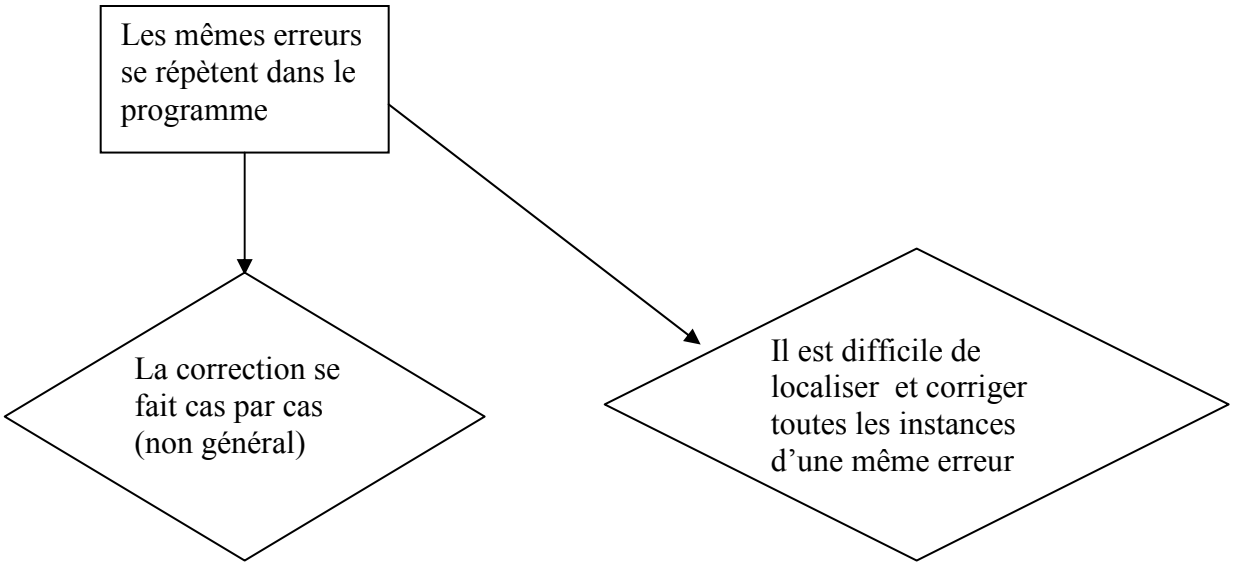
-le code peut être réutilisé avec un minimum d'effort.
-les développeurs corrigent chaque bogue au cas par cas sans définir une méthode générale de résolution.

Structure du Cut-and-Paste Programming

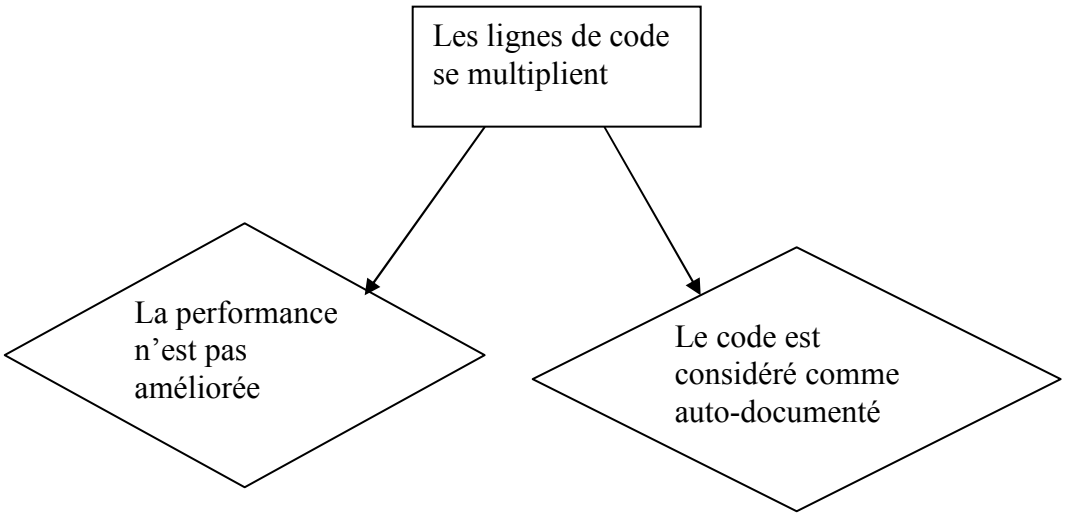
→ Niveau de la conception

S/O

→ Niveau flot d'exécution



→ Niveau de la classe



→ Niveau de la méthode

S/O

II-2) Les mini-anti-patrons

II-2.1) Le Continuous Obsolescence

La technologie avance tellement rapidement que les développeurs ont du mal à continuer avec les versions courantes des logiciels et avec les combinaisons de logiciels qui vont ensemble. On assiste donc a la sortie d'une nouvelle version dans un court lapse de temps.

II-2.2) Le Boat Anchor

Le Boat Anchor est une partie du code qui n'a aucun objectif ni aucun rôle dans le projet. Le Boat Anchor est détectable mais pas facilement.

II-2.3) Le Dead End

Le Dead End est observé lorsqu'on modifie du code réutilisable si ce code n'est plus maintenu ni supporté par le fournisseur. Les améliorations du code réutilisé ne sont pas facilement faisables et les problèmes de soutient sont blâmés par la modification. Il n'est pas détectable facilement.

II-2.4) Le Input Kludge

Le Input Kludge est facilement détectable lorsque le logiciel échoue à des tests comportementaux. Par exemple, lorsque le programme permet à l'utilisateur de faire n'importe quelle opération pouvant entraîner un crash ou une mauvaise utilisation du logiciel comme permettre à l'usager d'entrer une date dont le jour est supérieur a 31.

II-2.5) Le Walking through a Mine Field

Le Walking through a Mine Field apparaît lorsque les programmeurs sont convaincus que leur logiciel fonctionne à la perfection jusqu'au moment ou confrontés aux systèmes d'exploitations ou a d'autres supports, on observe de nombreux bogues jusque la jamais observés. Il est surtout détectable lorsqu'il y a des commentaires.

III-2.6) Le Mushroom Management

Le Mushroom Management est observé lorsque les développeurs et les utilisateurs finaux ne sont pas en contact. Il existe donc des personnes qui servent d'intermédiaires. Cette gestion entraîne plusieurs problèmes. Généralement les demandes de changement se font environs à 30% des coûts de développement. Dans un projet avec Mushroom Management les développeurs n'étant pas en contact avec les usagers lors de la conception, ils ne sont au courant des changements potentiels que lors de la livraison et cela peut entrainer de forts coûts de développement. Il n'est pas détectable.

III DETECTION DES ANTI-PATRONS DANS NUTCH 0.7.1

III-1 Technique de recherche des anti-patrons

La technique choisie pour trouver les anti-patrons dans le code consiste à passer en revue chaque classe une par une puis à déterminer pour chacune d'entre elle tous les anti-patrons qu'on y retrouve.

L'objectif étant de vérifier les résultats obtenus grâce à un outil, il fallait que traverser chaque classe. Le projet ayant été ouvert dans **éclipse** et celui-ci proposant une hiérarchie, le parcourt des classes a été plus facile.

Au début, n'étant pas encore très a l'aise avec la notion d'anti-patrons, la technique que j'utilisais était la suivante : grâce à la structure de chaque anti-patron que j'avais établie, j'entrais dans chaque classe et pour chaque type d'anti-patron, je cherchais sa structure dans le code. Cette technique était longue mais sûre.

Après avoir travaillé sur une cinquantaine de classe, j'étais plus familière avec les anti-patrons et lorsque j'entrais dans une classe j'étais directement en mesure de les détecter sans avoir à les traiter au cas par cas.

III-2 Recherche détaillée dans Nutch 0.7.1

III-2 Recherche dans src/java

III-2.1 package 'org.apache.nutch.analysis'

1. Interface CharStream.java
   -->pas d'anti-patrons
2. Classe CommonGrams.java
   -->pas d'anti-patrons
3. Classe FastCharStream.java
   -Spaghetti Code -->plusieurs fonctions n'ont pas de paramètres utilisent les variables globales.
4. Classe NutchAnalysis.java
   -Blob-->dans le design on a une grande classe centrale, 1 classe de donnée, une classe vide.
   De plus la classe contient plus de 60 attributs

-Lava Flow -->grand bout de code non commenté dont on ne sait pas le rôle

-Functional Decomposition--> des syntaxes comme enable_tracing ou jj_add_error_token...

Poltergeist-->on a des classes sans état dont on ne connait le but comme JJcalls et chemins superflus

-Spaghetti Code --> plusieurs méthodes objets ne contiennent pas de paramètres et des méthodes nommées comme des processus et il ya plusieurs switch et if/else (agit comme des goto)

5. Classe NutchAnalysisConstant.java
   -->pas d'anti-patrons.
6. Classe NutchAnalysisTokenManager.java
   Lava Flow--> code pas documenté du tout
   Spaghetti Code--> usage excessif de switch
7. Classe NutchDocumentAnalyzer.java
   -Functional Decomposition-->certaines classes ne font qu'une seule chose comme avoir une fonction.
   Poltergeist-->les classes apparaisse brièvement pour en initialiser d'autres, leur cycle est court.
8. Classe NutchDocumentTokenizer.java
   Pas d'anti patrons
9. Classe ParseException.java
   -Functional Decomposition-->nom de fonction comme add_escapes<
10. Classe Token.java
    Pas d'anti-patrons
11. Interface TokenManager.java
    Pas d'anti-patrons
12. Classe TokenMgrError.java
    Pas d'anti-patrons

III-2.2 package 'org.apache.nutch.clustering'

1. Interface HitsCluster.java
   -->aucun anti-patron
2. Interface OnlineClusterer.java
   -->aucun anti-patron
3. Classe OnlineClustererFactory.java
   -->aucun anti-patron

III-2.3 package org.apache.nutch.db

1. Classe BucketSet.java
   -->aucun anti-patron
2. Classe DBKeyDivision.java
   -->aucun anti-patron
3. Classe DBSectionReader.java
   -->aucun anti-patron
4. Classe DistributedWebDBReader.java
   -Functional Decomposition -->plusieurs classes qui ne font pas grand chose comme juste appeler une fonction.
   - Poltergeist.
5. Classe DistributedWebDBWriter.java
   -Blob-->on a plus de 60 attributs.
6. Classe EditSectionGroupReader.java
   Functional Decomposition-nom de fonction
   Spaghetti-->nom de fonction comme delete
7. Classe EditSectionGroupWriter.java
   Poltergeist car plusieurs classes n'apparaissent que pour en introduire d'autres et leurs cycle de vie est bref.
   -Functional Decomposition comme close
   Spaghetti-->nom de fonction comme close
8. Classe EditSectionWriter.java
   Functional Decomposition ou fonction comme append, close
   Spaghetti-->nom de fonction comme append, close
9. Interface IWebDBReader.java
   Pas d'anti-patrons (interface)
10. Interface IWebDBWriter.java
    Pas d'anti-patrons
11. Classe Link.java
    -cut and paste programming pour les classes internes.
12. Classe Page.java
    -Spaghetti Code -->plusieurs fonctions ne possèdent pas de paramètres et utilisent toutes les variables globales.
13. Classe WebDBAnchors.java
    Pas d'anti-patrons
14. Classe WebDBInjector.java
    Pas d'anti-patrons
15. Classe WebDBReader.java

Pas d'anti-patrons

16. Classe WebDBWriter.java

-Blob car on a plus de 60 attributs dans la classe

III-2.4 package org.apache.nutch.fetcher

1. Classe Fetcher.java

-Functional Decomposition--> nom de fonction comme run()

-Spaghetti car plusieurs fonctions sont sans paramètres et utilisent toutes les variables globales et nom de fonction, nom de fonction comme run()

2. Classe FetcherOutput.java

Pas d'anti-patrons

III-2.5.package org.apache.nutch.fs

1. Classe FileUtil.java

Pas d'anti-patrons

2. Classe LocalFileSystem.java

-Functional Decomposition , nom des fonctions

Spaghetti-->nom des fonctions

3. Classe NDFSFile

Pas d'anti-patrons

4. Classe NFSDataInputStream.java

Pas d'anti-patrons

5. Classe NFSDataOutputStream.java

Pas d'anti-patrons

6. Classe NFSInputStream.java

Pas d'anti-patrons

7. Classe NFSOutputStream.java

Pas d'anti-patrons

8. Classe NutchFileSystem.java

-Lava Flow-->certaines fonctions ne sont pas implémentées :on rencontre des parties "a faire"

-Functional Decomposition, + nom des fonctions

Spaghetti-->nom des fonctions

9. Classe TestClient.java

-Functional Decomposition

Spaghetti-->nom des fonctions

III-2.6.package org.apache.nutch.html

1. Classe Entities.java

Pas d'anti-patrons

III-2.7.package org.apache.nutch.indexer

1. Classe DeleteDuplicates.java
   -Functional Decomposition,
   Spaghetti-->nom des fonctions (composés aussi)
2. Classe HighFreqTerms.java
   Pas d'anti-patrons
3. Classe IndexingException.java
   Pas d'anti-patrons
4. Interface IndexingFilter.java
   Pas d'anti-patrons
5. Classe IndexingFilters.java
   Pas d'anti-patrons
6. Classe IndexMerger.java
   Pas d'anti-patrons
7. Classe IndexOptimizer.java
   -Lava Flow-->le code n'est pas documenté du tout.
8. Classe IndexSegment.java
   Pas d'anti-patrons
9. Classe NutchSimilarity.java
   Pas d'anti-patron

III-2.8.package org.apache.nutch.io

1. Classe ArrayFile.java
   Pas d'anti-patrons
2. Classe ArrayWritable.java
   Pas d'anti-patrons
3. Classe BooleanWritable.java
   Pas d'anti-patrons
4. Classe BytesWritable.java
   Pas d'anti-patrons
5. Classe DataInputBuffer.java
   Pas d'anti-patrons
6. Classe DataOutputBuffer.java
   Cut and Paste Programming (voir FloatWritable.java)
7. Classe FloatWritable.java
   Cut and Paste Programming (voir DataOutputBuffer.java)
8. Classe LongWritable.java
   Cut and Paste Programming (voir les deux classes precedente)

8**.clases IntWritable.java

Cut and Paste Programming (voir les deux classes precedente)

9. Classe MapFile.java
   Pas d'anti-patrons
10. Classe MD5Hash.java
   Pas d'anti-patrons
11. Classe NullWritable.java
   Pas d'anti-patrons
12. Classe SequenceFile.java
   -Functional Decomposition ,
   Spaghetti-->nom des fonctions
13. Classe SetFile.java
   Pas d'anti-patrons
14. Classe TwoDArrayWritable.java
   Pas d'anti-patrons
15. Classe UTF8.java
   -Lava Flow car on a du code non-utilisé laissé dans le programme
16. Classe VersionedWritable.java
   Pas d'anti-patrons
17. Classe VersionMismatchException.java
   Pas d'anti-patrons
18. Interface Writable.java
   Pas d'anti-patrons
19. Interface WritableComparable.java
   Pas d'anti-patrons
20. Classe WritableComparator.java
   Pas d'anti-patrons
21. Classe WritableName.java
   Pas d'anti-patrons
22. Classe WritableUtils.java
   Pas d'anti-patrons

III-2.8.package org.apache.nutch.ipc

1. Classe Client.java
   -Functional Decomposition → nom des fonctions
   Spaghetti-->nom des fonctions (stop)
2. Classe RPC.java
   Pas d'anti-patrons
3. Classe Server.java

- Functional Decomposition ,
Spaghetti-->nom des fonctions (stop)

III-2.9.package org.apache.nutch.linkdb

1. Classe LinkAnalysisEntry.java
Pas d'anti-patrons

III-2.10.package org.apache.nutch.mapReduce

1. Classe CombiningCollector.java
Pas d'anti-patrons
2. Interface Configurable.java
Pas d'anti-patrons
3. Classe FileSplit.java
Pas d'anti-patrons
4. Interface InputFormat.java
Pas d'anti-patrons
5. Classe InputFormatBase.java
Pas d'anti-patrons
6. Classe InputFormats.java
Pas d'anti-patrons
7. Interface InterTrackerProtocol.java
Pas d'anti-patrons
8. Classe JobClient.java
Pas d'anti-patrons
9. Classe JobConf.java
Pas d'anti-patrons
10. Classe JobProfile.java
Pas d'anti-patrons
11. Classe JobStatus.java
Pas d'anti-patrons
12. Interface JobSubmissionProtocol.java
Pas d'anti-patrons
13. Classe JobTracker.java
Blob-->nombreux attributs
14. Classe JobTrackerInfoServer.java
-Functional Decomposition nom des fonctions
Spaghetti-->nom des fonctions (stop, start)
15. Classe MapOutputFile.java
Pas d'anti-patrons
16. Classe MapOutputLocation.java

Pas d'anti-patrons

17. Interface MapOutputProtocol.java
    Pas d'anti-patrons
18. Interface Mapper.java
    Pas d'anti-patrons
19. Classe MapTask.java
    Pas d'anti-patrons
20. Classe MapTaskRunner.java
    Pas d'anti-patrons
21. Interface MRConstants.java
    Pas d'anti-patrons
22. Interface OutputCollector.java
    Pas d'anti-patrons
23. Interface OutputFormat.java
    Pas d'anti-patrons
24. Classe OutputFormats.java
    Pas d'anti-patrons
25. Interface Partitioner.java
    Pas d'anti-patrons
26. Interface RecordReader.java
    Pas d'anti-patron
27. Interface RecordWriter.java
    Pas d'anti-patrons
28. Interface Reducer
    Pas d'anti-patrons
29. Classe ReduceTask.java
    Pas d'anti-patrons
30. Classe ReduceTaskRunner.java
    Pas d'anti-patrons
31. Interface RunningJob.java
    Pas d'anti-patrons
32. Classe SequenceFileInputFormat.java
    Pas d'anti-patrons
33. Classe SequenceFileOutputFormat.java
    Pas d'anti-patrons
34. Classe Task.java
    Pas d'anti-patrons
35. Classe TaskRunner.java

Pas d'anti-patrons

36. Classe TaskStatus.java

Pas d'anti-patrons

37. Classe TaskTracker.java

Pas d'anti-patrons

38. Classe TaskTrackerStatus.java

Pas d'anti-patrons

39. Interface TaskUmbilicalProtocol.java

Pas d'anti-patrons

40. Classe TextInputFormat.java

Pas d'anti-patrons

41. Classe TextOutputFormat.java

Pas d'anti-patrons

III-2.11.package org.apache.nutch.mapReduce.demo

1. Classe Grep.java

Pas d'anti-patrons

III-2.12.package org.apache.nutch.mapReduce.lib

1. Classe HashPartitioner.java

Pas d'anti-patrons

2. Classe IdentityMapper.java

Pas d'anti-patrons

3. Classe IdentityReducer.java

Pas d'anti-patrons

4. Classe InverseMapper.java

Pas d'anti-patrons

5. Classe LongSumReducer.java

Pas d'anti-patrons

6. Classe RegexMapper.java

Pas d'anti-patrons

7. Classe TokenCountMapper.java

Pas d'anti-patrons

III-2.13.package org.apache.nutch.ndfs

1. Classe Block.java

Pas d'anti-patrons

2. Classe DatanodeInfo.java

Spaghetti Code--->plusieurs méthodes n'ont pas de paramètres et utilisent les variables globales

3. Classe DF.java

-Walking through a Mine Field-->le code ne marche que sous linux

4. Interface FSConstants.java

Lava Flow---^>parties a corriger a supprimer ou a faire

5. Classe FSDataset.java

Pas d'anti-patrons

6. Classe FSDirectory.java

-Functional Decomposition,

Spaghetti-->nom des fonctions (stop start)

7. Classe FSNamesystem.java

BLOB-->plus de 60 attributs

Functional Decomposition-->classe qui ne fait qu'avoir une fonction

Lava Flow-->interface inutilisé laissé dans le code

8. Classe FSParam.java

Pas d'anti-patrons

9. Classe FSResults.java

Pas d'anti-patrons

10. Classe HeartbeatData.java

Pas d'anti-patrons

11. Classe NDFS.java

Functional Decomposition-->classe n'ayant qu'une fonction

Spaghetti Code-->utilisation excessive du switch

12. Classe NDFSClient.java

Functional Decomposition-->classe n'ayant qu'une fonction + nom des fonctions

Spaghetti-->nom des fonctions (delete create)

13. Classe NDFSFile.java

Pas d'anti-patrons

14. Classe NDFSFileInfo.java

Pas d'anti-patrons

III-2.13.package org.apache.nutch.net

1. Classe BasicUrlNormalizer.java

-Blob-->une classe principale et une petite classe ne contenant que des données.

2. Classe RegexUrlNormalizer.java

-Blob-->une classe principale et une petite classe ne contenant que des données.

3. Interface URLFilter.java
Pas d'anti-patrons
4. Classe URLFilterChecker.java
Pas d'anti-patrons
5. Classe URLFilterException.java
Pas d'anti-patrons
6. Classe URLFilters.java
Pas d'anti-patrons
7. Interface UrlNormalizer.java
Pas d'anti-patrons
8. Classe UrlNormalizerFactory.java
Pas d'anti-patrons

III-2.14.package org.apache.nutch.net.protocols
1. Classe HttpDateFormat.java
Pas d'anti-patrons
2. Classe ProtocolException.java
Pas d'anti-patrons
3. Interface Response.java
Pas d'anti-patrons

III-2.14.package org.apache.nutch.ontology
1. Interface Ontology
Lava Flow-->parties ' à implémenter'
2. Classe OntologyFactory.java
Pas d'anti-patrons

III-2.15.package org.apache.nutch.paged
1. Classe FetchListEntry.java
Spaghetti Code--> plusieurs fonctions sans parametres utilisent les variables globales.
III-2.16.package org.apache.nutch.parse
1. Classe HTMLMetaTags.java
Pas d'anti-patrons
2. Interface HtmlParseFilter.java
Pas d'anti-patrons
3. Classe HtmlParseFilters.java
Pas d'anti-patrons

4. Classe Outlink.java
Pas d'anti-patrons
5. Classe OutlinkExtractor.java
Pas d'anti-patrons
6. Interface Parse.java
Pas d'anti-patrons
7. Classe ParseData.java
Pas d'anti-patrons
8. Classe ParseException.java
Pas d'anti-patrons
9. Classe ParseImpl.java
Pas d'anti-patrons
10. Interface Parser.java
Pas d'anti-patrons
11. Classe ParserChecker.java
Pas d'anti-patrons
12. Classe ParserFactory.java
Pas d'anti-patrons
13. Classe ParserNotFound.java
Pas d'anti-patrons
14. Classe ParseStatus.java
Pas d'anti-patrons
15. Classe ParseText.java
Pas d'anti-patrons

III-2.17.package org.apache.nutch.plugin
1. Classe Extension.java
Pas d'anti-patrons
2. Classe ExtensionPoint.java
Pas d'anti-patron
3. Classe Plugin.java
Pas d'anti-patrons
4. Classe PluginClassLoader.java
Pas d'anti-patrons
5. Classe PluginDescriptor.java
Pas d'anti-patron
6. Classe PluginManifestParser.java

Pas d'anti-patrons

7. Classe PluginRepository.java

Pas d'anti-patrons

8. Classe PluginRuntimeException.java

Pas d'anti-patrons

III-2.18.package org.apache.nutch.protocol

1. Classe Content.java

Lava Flow-->code non documenter on ne sait pas exactement ce que fait la classe.

2. Interface Protocol.java

Pas d'anti-patrons

3. Classe ProtocolException.java

Pas d'anti-patrons

4. Classe ProtocolFactory.java

Pas d'anti-patrons

5. Classe ProtocolNotFound.java

Pas d'anti-patron

6. Classe ProtocolOutput.java

Pas d'anti-patron

7. Classe ProtocolStatus.java

Pas d'anti-patron

8. Classe ResourceGone.java

Pas d'anti-patron

9. Classe ResourceMoved.java

Pas d'anti-patron

10. Classe RetryLater.java

Pas d'anti-patron

III-2.19.package org.apache.nutch.quality.dynamic

1. Classe PageDescription.java

Lava Flow-->code non documenté

Spaghetti Code-->usage excessif de switch (agissant comme des goto)

2. Interface PageDescriptionConstants.java

Pas d'anti-patron

3. Classe PageDescriptionTokenManager.java

Lava Flow-->code long et non-documenté

Spaghetti Code-->usage excessif de switch (agissant comme des goto

4. Classe ParseException.java
Pas d'anti-patron
5. Classe SimpleCharStream.java
Pas d'anti-patron
6. Classe Token.java
Pas d'anti-patron
7. Classe TokenMgrError.java
Pas d'anti-patron
III-2.20.package org.apache.nutch.searcher
1. Classe DistributedSearch.java
Pas d'anti-patron
2. Classe FetchedSegments.java
Pas d'anti-patron
3. Classe FieldQueryFilter.java
Pas d'anti-patron
4. Classe Hit.java
Pas d'anti-patron
5. Interface HitContent.java
Pas d'anti-patron
6. Interface HitDetailer.java
Pas d'anti-patron
7. Classe HitDetails.java
Pas d'anti-patron
8. Classe Hits.java
Pas d'anti-patron
9. Interface HitSummarizer.java
Pas d'anti-patron
10. Classe IndexSearcher.java
Pas d'anti-patron
11. Classe LuceneQueryOptimizer.java
Pas d'anti-patron
12. Classe NutchBean.java
Functional Decomposition-->classe n'ayant qu'une fonction
13. Classe OpenSearchServlet.java
Pas d'anti-patron
14. classe Query.java
Pas d'anti-patron
15. Classe QueryException.java

Functional Decomposition-->classe n'ayant qu'une fonction

16. Interface QueryFilter.java

Pas d'anti-patron

17. Classe QueryFilters.java

Pas d'anti-patron

18. Classe RawFieldQueryFilter.java

Pas d'anti-patron

19. Interface Searcher.java

Pas d'anti-patron

20. Classe Summarizer.java

-Lava Flow -->il y a des sections a remplacer

21. Classe Summary.java

-Functional Decomposition, nom des fonctions

Spaghetti-->nom des fonctions (stop start)

III-2.20.package org.apache.nutch.segment

1. Classe SegmentReader.java

-Functional Decomposition,

Spaghetti-->nom des fonctions (stop start)

2. Classe SegmentSlicer.java

Pas d'anti-patron

3. classe SegmentWriter.java

-Functional Decomposition,

Spaghetti-->nom des fonctions (close)

III-2.21.package org.apache.nutch.servlet

1. Classe Cached.java

Pas d'anti-patron

III-2.22.package org.apache.nutch.tools

1. Classe CrawlTool.java

Pas d'anti-patron

2. Classe DistributedAnalysisTool.java

-Functional Decomposition, nom des fonctions

Spaghetti-->nom des fonctions (stop start)

3. Classe FetchListTool.java

-Functional Decomposition, nom des fonctions

Spaghetti-->nom des fonctions (stop start)

4. Classe LinkAnalysisTool.java

Pas d'anti-patron

5. Classe ParseSegment.java

Pas d'anti-patron

6. Classe PruneIndexTool.java

Pas d'anti-patron

7. Classe SegmentMergeTool.java

Poltergeist-->chemins de navigation complexes

8. Classe UpdateDatabaseTool.java

Pas d'anti-patron

9. Classe UpdateSegmentsFromDb.java

Functional Decomposition-->l classe avec une seule fonction

10. Classe WebDBAdminTool.java

Pas d'anti-patron

III-2.23.package org.apache.nutch.util

1. Classe CommandRunner.java

+-Functional Decomposition-->des classes qui ne font qu'une chose

-Lava Flow-->code non commenté

2. Classe Daemon.java

Pas d'anti-patron

3. Classe FibonacciHeap.java

-Functional Decomposition-->nom des fonctions

sPAGHETTI -->nom des fonctions

4. Classe GZIPUtils.java

Pas d'anti-patron

5. Classe LogFormatter.java

Pas d'anti-patron

6. Classe NutchConf.java

Pas d'anti-patron

7. Classe PrefixStringMatcher.java

Pas d'anti-patron

8. Classe ScoreStats.java

Pas d'anti-patron

9. Classe StringUtil.java

Pas d'anti-patron

10. Classe SuffixStringMatcher.java

Pas d'anti-patron

11. Classe ThreadPool.java

Pas d'anti-patron

12. Classe TrieStringMatcher.java

Pas d'anti-patron

III-2.24.package org.apache.nutch.util.mime
1. Classe MimeType.java
-Functional Decomposition ,
Spaghetti-->nom des fonctions (stop start)
2. Classe MimeTypeException.java
Pas d'anti-patrons
3. Classe MimeTypes.java
-Lava Flow-->On rencontre des sections a 'remplacer'
4. Classe MimeTypesReader.java
Pas d'anti-patrons
III-3.Recherche dans src/plugin/clustering-carrot2/src/java
III-3.1.package org.apache.nutch.clustering.carrot2
1. Classe Clusterer.java
Lava Flow-->sections 'a remplacer'
2. Classe HitsClusterAdapter.java
Pas d'anti-patrons
3. Classe LocalNutchInputComponent.java
Pas d'anti-patrons
4. Classe NutchDocument.java
Pas d'anti-patrons
III-4.Recherche dans src/plugin/clustering-carrot2/src/test
III-4.1.package org.apache.nutch.clustering.carrot2
1. Classe ClustererTest.java
Pas d'anti-patrons
III-5.Recherche dans src/plugin/creativecommons/src/java
III-5.1.package org.creativecommons.nutch
1. Classe CCDeleteUnlicensedTool.java
Functional Decomposition-->nom des fonctions comme close()
spaghetti-->nom des fonctions comme close()
2. Classe CCIndexingFilter.java
Pas d'anti-patrons
3. Classe CCParseFilter.java
Pas d'anti-patrons
4. Classe CCQueryFilter.java
Pas d'anti-patrons
III-6.Recherche dans src/plugin/creativecommons/src/test
III-6.1.package org.creativecommons.nutch
1. Classe TestCCParseFilter.java

-Lava Flow --> discutable car le code n'est pas commenté du tout et on ne sait pas ce qu'il fait

III-7.Recherche dans src/plugin/index-basic/src/java
III-7.1.package org.apache.nutch.indexer.basic
1. Classe BasicIndexingFilter.java
Pas d'anti-patrons
III-8.Recherche dans src/plugin/index-more/src/java
III-8.1.package org.apache.nutch.indexer.more
1. Classe MoreIndexingFilter.java
Pas d'anti-patrons
III-9.Recherche dans src/plugin/languageidentifier/src/java
III-9.1.package org.apache.nutch.analysis.lang
1. Classe HTMLLanguageParser.java
Pas d'anti-patrons
2. Classe LanguageIdentifier.java
Pas d'anti-patrons
3. classe LanguageIndexingFilter.java
Pas d'anti-patrons
4. Classe LanguageQueryFilter.java
Pas d'anti-patrons
5. Classe NGramProfile.java
-Lava Flow-->on a des parties "à faire''
III-10.Recherche dans src/plugin/languageidentifier/src/test
III-10.1.package org.apache.nutch.analysis.lang
1. Classe TestHTMLLanguageParser.java
Pas d'anti-patrons
2. Classe TestLanguageIdentifier.java
Pas d'anti-patrons
3. Classe TestNGramProfile.java
Pas d'anti-patrons
III-10*.Recherche dans src/plugin/ontology/src/java
III-10.1.package org.apache.nutch.ontology
1. Classe OntologyImpl.java
-Lava Flow-->parties`a faire` et plusieurs blocs non commentés
2. Classe OwlParser.java
Pas d'anti-patrons
3. Interface Parser.java

Pas d'anti-patrons

III-11.Recherche dans src/plugin/ontology/src/test

III-11.1.package org.apache.nutch.ontology

1. Classe TestOntology .java

Pas d'anti-patrons

III-12.Recherche dans src/plugin/parse-ext/src/java

III-12.1.package org.apache.nutch.parse.ext

1. Classe ExtParser.java

Pas d'anti-patron

III-13.Recherche dans src/plugin/parse-ext/src/test

III-13.1.package org.apache.nutch.parse.ext

1. Classe TestExtParser.java

Walking through a Mine Field-->le code ne marche que sous linux

III-13*.Recherche dans src/plugin/parse-html/src/java

III-13.1.package org.apache.nutch.parse.html

1. Classe DOMBuilder.java

-Lava Flow-->plusieurs fonctions pas encore implémentées mais gardées dans le code en prévision de quelque chose

2. Classe DOMContentUtils.java

Pas d'anti-patrons

3. Classe HTMLMetaProcessor.java

Poltergeist--->chemin de navigation

4. Classe HtmlParser.java

Pas d'anti-patrons

5. Classe XMLCharacterRecognizer.java

Pas d'anti-patrons

III-14.Recherche dans src/plugin/parse-html/src/test

III-14.1.package org.apache.nutch.parse.html

1. Classe TestDOMContentUtils.java

Pas d'anti-patrons

2. Classe TestRobotsMetaProcessor.java

Pas d'anti-patrons

III-15.Recherche dans src/plugin/parse-js/src/java

III-15.1.package org.apache.nutch.parse.js

1. Classe JSParseFilter.java

Pas d'anti-patrons

III-16.Recherche dans src/plugin/parse-mp3/src/java

III-16.1.package org.apache.nutch.parse.mp3

1. Classe MetadataCollector.java
Pas d'anti-patrons
2. Classe MP3Parser.java
Pas d'anti-patrons
III-17.Recherche dans src/plugin/parse-mp3/src/test
III-17.1.package org.apache.nutch.parse.mp3
1. Classe TestMP3Parser.java
Pas d'anti-patrons

III-18.Recherche dans src/plugin/parse-msword/src/java
III-18.1.package org.apache.nutch.parse.msword
1. Classe FastSavedException.java
Pas d'anti-patrons
2. Classe MSWordParser.java
Pas d'anti-patrons
3. Classe PasswordProtectedException.java
Pas d'anti-patrons
4. Classe Test.java
Pas d'anti-patrons
5. Classe Word6Extractor.java
Pas d'anti-patrons
6. Classe WordExtractor.java
Pas d'anti-patrons
7. Classe WordTextBuffer.java
Pas d'anti-patrons
8. Classe WordTextPiece.java
Pas d'anti-patrons
III-18.2.package org.apache.nutch.parse.msword.chp
1. Classe Word6CHPBinTable.java
Pas d'anti-patrons
III-19.Recherche dans src/plugin/parse-msword/src/test
III-19.1.package org.apache.nutch.parse.msword
1. Classe TestMSWordParser.java
Pas d'anti-patrons
III-20.Recherche dans src/plugin/parse-pdf/src/java
III-20.1package org.apache.nutch.parse.pdf
1. Classe PdfParser.java
Pas d'anti-patrons

III-21.Recherche dans src/plugin/parse-rss/src/java
III-21.1.package org.apache.nutch.parse.rss
1. Classe FeedParserListenerImpl.java
Pas d'anti-patrons
2. Classe RSSParser.java
Pas d'anti-patrons
III-21.2.package org.apache.nutch.parse.rss.structs
1. Classe RSSChannel.java
Pas d'anti-patrons
2. Classe RSSItem.java
Pas d'anti-patrons
III-22.Recherche dans src/plugin/parse-rss/src/test
III-22.1.org.apache.nutch.parse.rss
1. Classe TestRSSParser.java
Pas d'anti-patrons
III-23.Recherche dans src/plugin/parse-rtf/src/java
III-23.1.package org.apache.nutch.parse.rtf
1. Classe RTFParseFactory.java
Pas d'anti-patrons
2. Classe RTFParserDelegateImpl.java
-Lava Flow--> le code contient des parties 'à faire'
Functional Decomposition
III-23.Recherche dans src/plugin/parse-rtf/src/test
III-23.1.package org.apache.nutch.parse.rtf
1. Classe TestRTFParser.java
Pas d'anti-patrons
III-24.Recherche dans src/plugin/parse-text/src/java
III-24.1.package org.apache.nutch.parse.text
1. Classe TextParser.java
-Lava Flow-->section 'a corriger'
III-25.Recherche dans src/plugin/protocol-file/src/java
III-25.1.package org.apache.nutch.protocol.file
1. Classe File.java
Pas d'anti-patrons
2. Classe FileError.java
Pas d'anti-patrons
3. Classe FileException.java
Pas d'anti-patrons

4. Class FileResponse
-Lava Flow-->section 'à corriger'
III-26.Recherche dans src/plugin/protocol-ftp/src/java
III-26.1.package org.apache.nutch.protocol.ftp
1. Classe Client.java
-Lava Flow-->section 'à corriger'
Spaghetti Code
2. Classe Ftp.java
Pas d'anti-patrons
3. Classe FtpError.java
Pas d'anti-patrons
4. Classe FtpException.java
Pas d'anti-patrons
5. Classe FtpExceptionBadSystResponse.java
Pas d'anti-patrons
6. Classe FtpExceptionCanNotHaveDataConnection.java
Pas d'anti-patrons
7. Classe FtpExceptionControlClosedByForcedDataClose.java
Pas d'anti-patrons
8. Classe FtpExceptionUnknownForcedDataClose.java
Pas d'anti-patrons
9. Classe FtpResponse.java
-Lava Flow-->section 'a corriger'
Spaghetti Code-->abus excessif de if/else
10. Classe PrintCommandListener.java
Pas d'anti-patrons
III-27.Recherche dans src/plugin/protocol-http/src/java
III-27.1.package org.apache.nutch.protocol.http
1. Classe Http.java
Pas d'anti-patrons
2. Classe HttpError.java
Pas d'anti-patrons
3. Classe HttpException.java
Pas d'anti-patrons
4. Classe HttpResponse.java
Pas d'anti-patrons
5. Classe RobotRulesParser.java
Pas d'anti-patrons

III-27.Recherche dans src/plugin/protocol-http/src/test
III-27.1.package org.apache.nutch.protocol.http
1. Classe TestRobotRulesParser.java
Pas d'anti-patrons
III-27.Recherche dans src/plugin/protocol-httpclient/src/java
III-27.1.package org.apache.nutch.protocol.httpclient
1. Classe DummySSLProtocolSocketFactory.java
Pas d'anti-patrons
2. Classe DummyX509TrustManager.java
Pas d'anti-patrons
3. Classe Http.java
Pas d'anti-patrons
4. Interface HttpAuthentication.java
Pas d'anti-patrons
5. Classe HttpAuthenticationException.java
Pas d'anti-patrons
6. Classe HttpAuthenticationFactory.java
-Lava Flow-->section 'a faire'
7. Classe HttpBasicAuthentication.java
Pas d'anti-patrons
8. Classe HttpError.java
Pas d'anti-patrons
9. Classe HttpException.java
Pas d'anti-patrons
10. Classe HttpResponse.java
Pas d'anti-patrons
11. Classe MultiProperties.java
Pas d'anti-patrons
12. Classe RobotRulesParser.java
Pas d'anti-patrons

III-28.Recherche dans src/plugin/query-basic/src/java
III-28.1. Classe BasicQueryFilter.java
1. Classe BasicQueryFilter.java
Pas d'anti-patrons
III-29.Recherche dans src/plugin/query-more/src/java
III-29.1.package org.apache.nutch.searcher.more
1. Classe DateQueryFilter.java

Pas d'anti-patrons

2. Classe TypeQueryFilter.java

Pas d'anti-patrons

III-30.Recherche dans src/plugin/query-site/src/java

III-30.1.package org.apache.nutch.searcher.site

1. Classe SiteQueryFilter.java

Pas d'anti-patrons

III-30.Recherche dans src/plugin/query-url/src/java

III-30.1.package org.apache.nutch.searcher.url

1. Classe URLQueryFilter.java

Pas d'anti-patrons

III-31.Recherche dans src/plugin/urlfilter-prefix/src/java

III-31.1.package org.apache.nutch.net

1. Classe PrefixURLFilter.java

Pas d'anti-patrons

III-32.Recherche dans src/plugin/urlfilter-regex/src/java

III-32.1.package org.apache.nutch.net

1. Classe RegexURLFilter.java

Blob-->une classe centrale et une petite classe qui n'a que des données.

III-33.Recherche dans src/test

III-33.1.package org.apache.nutch.analysis

1. Classe TestQueryParser.java

Pas d'anti-patrons

III-33.2.package org.apache.nutch.db

1. Classe DBTester.java

Functional Decomposition a cause des nom de fonction (composées)

2. Classe TestWebDB.java

Pas d'anti-patrons

III-33.3.package org.apache.nutch.fetcher

1. Classe TestFetcher.java

Pas d'anti-patrons

2. Classe TestFetcherOutput.javaè

Pas d'anti-patrons

III-33.4.package org.apache.nutch.io

1. Classe RandomDatum.java

Pas d'anti-patrons

2. Classe TestArrayFile.java

-Lava Flow-->code pas assez documenté

3. Classe TestMD5Hash.java

Pas d'anti-patrons

4. Classe TestSequenceFile.java

-Lava Flow-->code complexe et pas documenté

5. Classe TestSetFile.java

-Lava Flow-->code pas assez documenté

6. Classe TestUTF8.java

Pas d'anti-patrons

7. Classe TestVersionedWritable.java

Pas d'anti-patrons

8. Classe TestWritable.java

Pas d'anti-patrons

III-33.5.package org.apache.nutch.ipc

1. Classe TestIPC.java

-Lava Flow-->code pas assez documenté

2. Classe TestRPC.java

Pas d'anti-patrons

III-33.6.package org.apache.nutch.net

1. Classe TestBasicUrlNormalizer.java

Pas d'anti-patrons

2. Classe TestRegexUrlNormalizer.java

Pas d'anti-patrons

III-33.7.package org.apache.nutch.pagedb

1. Classe TestFetchListEntry.java

Pas d'anti-patrons

2. Classe TestPage.java

Pas d'anti-patrons

III-33.8.package org.apache.nutch.parse
1. Classe TestOutlinkExtractor.java
Pas d'anti-patrons
2. Classe TestParseData.java
Pas d'anti-patrons
3. Classe TestParseText.java
Pas d'anti-patrons

III-33.9.package org.apache.nutch.plugin
1. Classe HelloWorldExtension.java
Functional Decomposition--> classe ne faisant qu'une chose:avoir une fonction
2. Interface ITestExtension.java
Pas d'anti-patrons
3. Classe SimpleTestPlugin.java
Pas d'anti-patrons
4. Classe TestPluginSystem.java
-Lava Flow-->code pas assez documenté
Spaghetti Code -->plusieurs fonctions ne possedent pas de parametres.

III-33.10.package org.apache.nutch.protocol
1. Classe TestContent.java
Pas d'anti-patrons

III-33.11.package org.apache.nutch.searcher
1. Classe TestHitDetails.java
-Lava Flow-->aucun commentaire
2. Classe TestQuery.java
-Lava Flow-->aucun commentaire
III-33.11.package org.apache.nutch.tools
1. Classe TestSegmentMergeTool.java
Spaghetti Code-->par les switch
III-33.12.package org.apache.nutch.util
1. classe TestFibonacciHeap.java
Pas d'anti-patrons

2. Classe TestGZIPUtils.java
Pas d'anti-patrons
3. Classe TestPrefixStringMatcher.java
Pas d'anti-patrons
4. Classe TestStringUtil.java
Pas d'anti-patrons
5. Classe TestSuffixStringMatcher.java
Pas d'anti-patrons

III-33.13.package org.apache.nutch.util.mime
1. Classe TestMimeType.java
Pas d'anti-patrons
2. Classe TestMimeTypes.java
-Lava Flow-->plusieurs parties a faire
Spaghetti Code--> plusieurs boucles imbriquées

III-3 Discussions

La recherche d'anti-patrons est un travail qui demande un peu plus que le fait de suivre à la lettre les caractéristiques répertoriées dans les ouvrages. En effet, l'analyste doit avoir en plus de la structure de chaque anti-patron un sens critique car la présence d'un anti-patron ou d'un autre est un sujet **très discutable**; il n'y a pas de règles fixes et on peut hésiter longtemps avant de décider si un certain anti-patron est présent ou non.

On remarque aussi que la présence de **certains types d'anti-patrons peut en entraîner d'autres** par exemple lorsqu'on observe un Spaghetti Code dû aux noms de fonctions (comme des processus), il y a de fortes chances qu'on observe aussi un Functional Decomposition.

Certains anti-patrons comme le Golden Hammer ou la plupart des mini-anti-patrons ne sont **pas détectables** surtout par une personne qui n'a pas participé au projet de conception pour les raisons suivantes :

- Pour détecter certains antis patron il faut par exemple savoir le choix d'outil que les programmeurs ont fait parmi plusieurs autres et la raison de ce choix.

- Quand on n'a pas vraiment participé à la programmation il quelque fois difficile sur des centaines de classes de savoir qu'un certain bout de code ne sert à rien et n'est pas utilisé.
- Certains anti-patrons sont dûs au fait que le programmeur ait recyclé du code qui n'est plus maintenu par les fournisseurs ; il est impossible de les détectés si on n'était pas la lors de l'implantation (sauf s'il y a des commentaires a ce sujet).
- On a aussi souvent besoin de savoir si les développeurs ont été en contact avec les utilisateurs finaux ou pas pour détecter certains anti-patrons.

III-4 Résultats obtenus

Suite à ma recherche dans Nutch 0.7.1, j'ai obtenue les résultats suivants pour 482 classes en tout.

Blob: 9

Lava Flow: 34

Poltergeist: 7

Functional Decomposition: 73

Spaghetti code: 56

Cut and paste programming: 5

Walking through a mine field: 2

Conclusion

J'ai trouvé ce projet sur lequel j'ai passé les quatre derniers mois très intéressant. Effectivement, il m'a donné une idée de la façon de faire de la recherche, du genre de travail qu'on peut avoir à faire en entreprise et en particulier, dans les laboratoires de recherche. Ce projet m'a surtout permis de savoir dans quel domaine j'aimerais le plus travailler plus tard.