UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAITRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
MARGUERITE FRANCINE DJOMBY

ASPECT MINING ET COHÉSION DES CLASSES: EXPLORATION DE L'UTILISATION DE LA COHÉSION POUR IDENTIFIER LES PRÉOCCUPATIONS TRANSVERSES

Décembre 2009

ASPECT MINING ET COHÉSION DES CLASSES:
EXPLORATION DE L'UTILISATION DE LA COHÉSION POUR IDENTIFIER LES PRÉOCCUPATIONS TRANSVERSES

Sommaire

La programmation orientée objet a été largement utilisée dans les milieux industriels. Elle a indéniablement fait avancer l'ingénierie des systèmes logiciels à différents points de vue. Cependant, les nombreuses retombées d'expériences ont révélé les limites sérieuses qu'elle éprouve dans le développement de systèmes logiciels complexes. Ces limites sont, entre autres, dues à la difficulté de prise en compte des préoccupations transverses (qui recoupent plusieurs parties d'un système). Ces préoccupations sont souvent dupliquées (et dispersées) dans le code. Ceci affecte la qualité des systèmes, en particulier au niveau de la modularité, et rend difficile, entre autres, leur compréhension ainsi que leur maintenance et évolution.

L'évolution des systèmes industriels complexes actuels est souvent inéluctable et présente des enjeux économiques et qualitatifs considérables. Une des solutions envisagées consiste à utiliser la programmation orientée aspect. Ce nouveau paradigme de programmation cherche, en effet, à améliorer la séparation (et l'intégration) des préoccupations en « modularisant » les éléments transversaux des systèmes sous forme d'unités séparées appelées « aspects ». La migration d'un système orienté objet vers un système orienté aspect passe d'abord par l'identification et la localisation de ces préoccupations dans le code objet (aspect mining) pour les représenter, par la suite, par des aspects (aspect refactoring). Dans ce projet, nous nous intéressons plus aux techniques d'identification des préoccupations transverses dans du code objet.

Nous avons exploré, comme nouvelle piste de recherche, l'utilisation de la « cohésion » (comparée aux techniques actuelles d'aspect mining) pour identifier les préoccupations transverses dans un code objet. Nous l'avons utilisée, en fait, comme indicateur de disparité dans le code, ce qui nous semblait être un symptôme de code pouvant correspondre à des préoccupations transverses. L'objectif de notre travail consistait essentiellement à vérifier cette hypothèse. Une étude expérimentale de grande envergure a été conduite, sur plusieurs systèmes logiciels Java, avec pour objectif d'explorer l'utilisation de la cohésion comme moyen de détection de classes potentielles contenant du code pouvant correspondre à des préoccupations transverses. Les résultats obtenus indiquent clairement que la cohésion pourrait être utilisée (moyennant éventuellement quelques ajustements) pour supporter l'aspect mining. Les corrélations obtenues dans le cas de plusieurs systèmes sont très fortes (entre la cohésion et les classes contenant des préoccupations transverses).

ASPECT MINING AND COHESION OF CLASSES:
EXPLORING THE USE OF COHESION TO IDENTIFY CROSSCUTTING CONCERNS

Abstract

The object-oriented programming paradigm has been widely used in industrial environments. It has been an important contribution to software engineering from various points of view. However, several studies revealed that the object paradigm suffers from serious limitations, particularly, in the development of large and complex software systems. These limitations are basically due to the difficulty in modularizing crosscutting concerns (which crosscut several parts of a system). The code corresponding to these concerns is often duplicated (and dispersed) in the program. This affects the quality of the program, particularly its modularity, and makes difficult among others its comprehension, maintenance and evolution.

The evolution of existing industrial software systems is often ineluctable and presents crucial qualitative and economic issues. One of the considered solutions consists in using the aspect-oriented programming. This new programming paradigm improves separation (and integration) of concerns by "modularizing" the elements that crosscut a system in separated units called "aspects". Migrating an object-oriented system towards an aspect-oriented system goes initially by the identification and the localization of these concerns in the object code (aspect mining), to represent them then by aspects (aspect refactoring). In this project, we are interested more in the identification techniques of crosscutting concerns in object-oriented code.

We explored a new research way using cohesion (compared to the current aspect mining techniques) to identify crosscutting concerns in an object-oriented code. We used it, in fact, as an indicator of disparity of the code, which seemed to us to be a symptom of code corresponding to crosscutting concerns. In this project, we wanted to validate this

hypothesis. We designed and conducted an experimental study on several Java software systems. The objective was to explore empirically the use of cohesion to detect potential classes containing code corresponding to crosscutting concerns. The obtained results indicate clearly that cohesion could be used (with possibly some adjustments) to support aspect mining. The obtained correlations, in the case of various systems, are very strong (between cohesion and the classes containing crosscutting concerns).

Remerciements

Je remercie

Mon Seigneur de m'avoir donnée la force et le courage pour pouvoir m'adapter dans mon nouvel environnement; et surtout de m'avoir couvert de toute sa protection et d'une excellente santé sans quoi je n'aurais pu bien mener à terme mon projet.

Mes directeurs de recherche Mourad Badri et Linda Badri pour leurs conseils, leur soutien intellectuel et financier. Votre grande disponibilité, votre amour pour le travail et votre abnégation indéniable sont pour moi un modèle. J'ai beaucoup appris durant ces années passées à vos côtés.

Mon père, mes frères et ma sœur de m'avoir soutenue moralement, intellectuellement et financièrement durant toutes mes études. Vos prières ont guidé mes pas et mes choix.

Je tiens à remercier particulièrement Olivier mon frère et son épouse Anne pour leur confiance. Leurs conseils, leur assistance et leur patience ont été un apport incommensurable durant tout mon chemin.

Mon fiancé Claude-Francis pour son soutien et son apport intellectuel indéniable, ses conseils réconfortants.

Tous mes amis pour leur disponibilité, particulièrement ceux du laboratoire de recherche en génie logiciel pour les beaux moments passés ensemble à apprendre, et surtout pour l'esprit fraternel qui y règne.

Ce mémoire est dédié à la mémoire de ma très chère mère, feue Ruth Essombé (1993-2009), il y a seize ans, tu nous quittas. Puisse le Seigneur te garder dans son glorieux royaume.

Ce projet a été rendu possible grâce à la contribution financière du CRSNG (Conseil de Recherches en Sciences Naturelles et en Génie du Canada) et à la fondation de l'UQTR.

Tables des matières

Liste des tableaux

Liste des Figures

CHAPITRE I

INTRODUCTION

De nos jours, l'un des soucis majeurs des développeurs est de promouvoir la qualité du logiciel. Le problème fondamental est de savoir comment produire des systèmes fiables, de haute qualité tout en réduisant les coûts de développement et de maintenance. La maintenance, en particulier, représente une phase très coûteuse du cycle de vie d'un logiciel [Sommerville 04]. Le paradigme orienté objet, grâce aux concepts qu'il a introduits, tels que l'encapsulation, le polymorphisme et l'héritage à fournit un excellent support au développement. Le principe fondamental de la conception objet recommande une forte cohésion et un faible couplage pour minimiser les dépendances entre les classes, maîtriser les changements et limiter leurs impacts. Cependant, les systèmes orientés objet éprouvent des difficultés sérieuses à cause de la présence des préoccupations transverses dans le code. Ces préoccupations sont très souvent dupliquées dans plusieurs endroits dans un programme; souvent dans une relation d'appel de méthode. Cette duplication (et dispersion) de code entrave certains critères fondamentaux de la qualité logicielle tels que la compréhensibilité, la réutilisabilité et la maintenabilité [Appel 07].

Dans [Sommerville 04], la compréhensibilité est définie comme la facilité allouée à un système pour permettre à l'utilisateur final de comprendre aisément les exigences ou les fonctionnalités. La réutilisabilité dans l'industrie du logiciel est le processus d'utilisation des modules existants sans avoir à les recréer [Basili 96]. Pour cela, un module doit être cohérent et autosuffisant à sa conception. La maintenabilité [Sommerville 04] est le degré de facilité alloué à un logiciel pour supporter les modifications. Les modifications incluent les corrections, les améliorations, l'adaptation aux changements dans l'environnement et aux exigences.

La présence de préoccupations transverses dans un code montre la limite de la modularité offerte par le paradigme orienté objet. La programmation orientée aspect vient combler ce gap, en ajoutant de nouveaux concepts permettant une meilleure modularisation. Ce paradigme permet d'améliorer certains facteurs de la qualité du logiciel, en séparant ces préoccupations. La migration d'un système orienté objet vers un système orienté aspect nécessite une recherche systématique de toutes ces préoccupations transverses dans un code objet qui peut être volumineux et complexe. Il devient alors nécessaire de développer des stratégies (et outils) permettant de détecter ces préoccupations : l'aspect mining constitue une approche. L'aspect mining, par ses techniques d'analyse de code, permet de supporter l'identification de ces préoccupations. Après identification de ces préoccupations, il faut pouvoir les modulariser grâce au paradigme orienté aspect.

I-1. Problématique

L'industrie du logiciel connait une croissance fulgurante de nos jours. Le développement de logiciel répond à plusieurs exigences (fonctionnelles et non fonctionnelles), et nécessite une bonne méthode de conception. Le développement orienté objet offre un excellent support grâce aux concepts qu'il a introduits. Les langages orientés objet éprouvent, cependant, des difficultés à représenter des préoccupations qui se recoupent dans un code. Ces préoccupations, souvent dupliquées et dispersées dans le code, sont appelées préoccupations transverses. Elles réduisent la qualité des programmes et augmentent leur complexité. Leur maintenance et évolution deviennent aussi difficiles. Une des solutions envisagées consiste à utiliser la programmation orientée aspect. Cette méthode de programmation permet de séparer et d'intégrer les préoccupations en modularisant celles qui sont transverses dans des unités séparées appelées aspects. Pour séparer ces préoccupations, il faut d'abord les identifier et les localiser. Dans ce projet, nous nous intéressons plus aux techniques d'identification des préoccupations transverses, en explorant une nouvelle piste de recherche basée sur l'utilisation de la

cohésion (comme indicateur de disparité dans le code). Ces préoccupations peuvent être, par exemple, la journalisation, la synchronisation ou une optimisation spécifique.

I-2. L'approche

Nous nous intéressons dans ce mémoire plus à l'identification des préoccupations transverses. S'appuyant sur les techniques d'aspect mining existantes, nous explorons une nouvelle piste de recherche utilisant la cohésion des classes (ou le manque de cohésion des classes). L'idée consiste à utiliser la cohésion, en fait, comme indicateur de disparité dans le code. La disparité dans un code peut être un symptôme de présence de préoccupations transverses. Notre objectif consiste à vérifier cette hypothèse. Nous avons mené une étude expérimentale de grande envergure sur plusieurs systèmes logiciels Java avec pour but d'examiner la possibilité d'utiliser la cohésion pour détecter les classes potentielles contenant du code pouvant correspondre à des préoccupations transverses. L'idée étant de déceler de façon automatique, par le biais de l'analyse statique du code objet, des symptômes de disparité dans le code. Pour valider notre approche, notre démarche a été, dans un premier temps, d'identifier les symptômes de disparité dans le code en utilisant une des techniques actuelles validées d'aspect mining (analyse fan-in). Dans un second temps, nous avons utilisé la cohésion pour déceler ces mêmes symptômes et nous avons comparé les résultats obtenus par les deux techniques. Dans cette partie, nous avons sélectionné plusieurs métriques de cohésion connues dans la littérature des métriques orientées objet. Plusieurs systèmes logiciels Java réels ont été considérés dans notre expérimentation. Pour analyser les données collectées et interpréter les résultats obtenus, nous avons utilisé la méthode de la régression logistique (logiciel XLSTAT). Nous avons, entre autres, évalué les corrélations entre les résultats fournis par l'outil d'aspect mining utilisé et ceux fournis en utilisant la cohésion des classes (plusieurs métriques). Les résultats obtenus sont concluants. Nous avons aussi effectué une étude comparative entre les métriques de cohésion évaluées.

I-3. Organisation

Ce mémoire est divisé en six chapitres : le deuxième chapitre porte sur l'état de l'art. Nous présentons d'abord les limites du paradigme objet, les solutions du paradigme aspect et enfin les différentes approches d'aspect mining existant dans la littérature. Le troisième chapitre aborde les métriques orientées objet en général, et les métriques de cohésion en particulier, question de ressortir les différentes améliorations apportées à l'évaluation de la cohésion des classes. Ceci permettra au lecteur de comprendre le choix porté sur certaines métriques et l'évaluation comparative qui en est effectuée dans le travail réalisé pour déterminer celles qui permettent de détecter le mieux les préoccupations transverses. L'approche adoptée est présentée au quatrième chapitre. Nous y présentons notre environnement de travail et les outils utilisés. La partie expérimentale de notre projet est présentée au cinquième chapitre. Elle résume la démarche utilisée et présente les systèmes et les métriques sélectionnées pour l'expérimentation. Les techniques utilisées pour l'extraction des données y sont définies, l'approche statistique aussi. Les résultats des expérimentations et leur interprétation vont clore cette partie. Nous concluons notre projet dans le sixième chapitre et ouvrons notre étude sur de potentielles pistes de recherche dérivées de notre travail.

CHAPITRE II
ÉTAT DE L'ART

II-1. Problématique

Les techniques de programmation ont évolué rapidement. Cependant, avec le développement de logiciels de plus en plus complexes, on a également eu besoin de langages plus performants, offrant des constructions (et concepts) permettant de supporter un développement de qualité à différents points de vue. La programmation objet a vu le jour dans ce contexte. Elle regroupe les données et les traitements dans un même composant appelé *classe*. Elle permet également d'implémenter facilement les spécifications fonctionnelles ou parties « métier » du logiciel. Cependant, elle éprouve des difficultés à bien implémenter des spécifications non fonctionnelles qui expriment les problèmes techniques du système. Ces fonctionnalités, encore appelées préoccupations transverses, concernent très souvent plusieurs modules. On peut citer, en exemple, la persistance des données, la journalisation, la sécurité ainsi que la gestion des exceptions. Ces préoccupations transverses impliquent très souvent un enchevêtrement du code. Ceci résulte, en fait, de l'implémentation de plusieurs problématiques ou besoins simultanés dans un même module. Elles impliquent également une dispersion du code: plusieurs modules du système peuvent être concernés par les mêmes préoccupations. La journalisation ainsi que la synchronisation, par exemple, peuvent concerner toutes les classes qui ont accès à une base de données dans le cas d'une application de gestion d'une base de données [Baltus 02]. Ces préoccupations sont implémentées dans plusieurs modules de l'application, ce qui crée une dispersion du code ou une duplication de celui-ci comme nous l'illustrons dans la figure 1 ci-après.

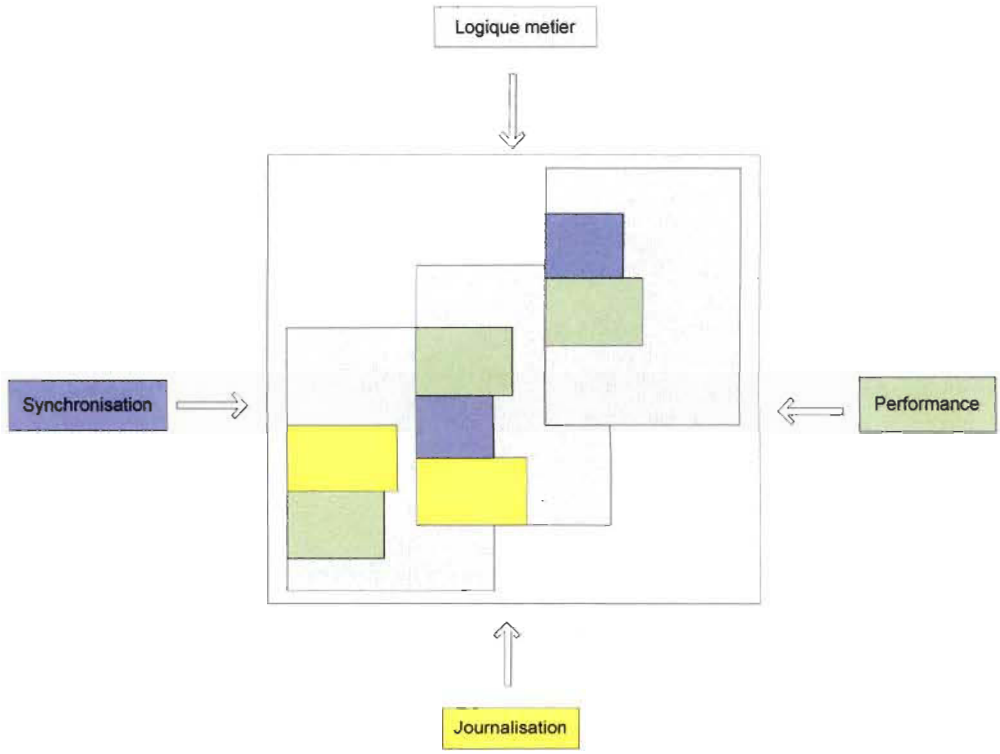


Figure 1: Recoupement des préoccupations dans les modules d'un système.

II-2. Les conséquences

Le recoupement des préoccupations dans le code objet impacte la qualité, la traçabilité, la réutilisation et la productivité de l'équipe de travail [Baltus 02].

Code de mauvaise qualité : L'enchevêtrement du code le rend très complexe et peut entrainer des anomalies difficiles à identifier. La dispersion des problématiques dans le code renforce la dépendance des modules (fort couplage), et engendre une augmentation de sa complexité.

Mauvaise traçabilité du code : Implémenter simultanément plusieurs préoccupations ne facilite pas leur identification. La correspondance entre ces préoccupations et leurs implémentations est peu claire.

Faible réutilisabilité du code : Avec un module qui implémente plusieurs responsabilités, il est peu probable qu'un autre système ait besoin de toutes ses fonctionnalités en même temps. La réutilisation de ce code nécessitera probablement une adaptation profonde au nouveau système.

En implémentant plusieurs préoccupations à la fois dans un même module, il est évident que le programmeur ne focalisera pas son attention seulement sur la préoccupation de base du module qui est le but premier de ce dernier. Il s'occupera aussi des aspects secondaires (besoins non-fonctionnels), ce qui peut réduire la productivité.

II-3. Solution

Une solution à ce problème est la programmation orientée aspect [Kiczales 97]. Cette technique de programmation factorise ou « modularise » les préoccupations transverses (besoins non-fonctionnels) qui se trouvent dispersées dans plusieurs endroits dans le code, en une entité plus facile à maitriser appelée *aspect*. Elle améliore la séparation des préoccupations et leur implémentation. Le paradigme orienté aspect se résume principalement en :

- L'identification et la séparation des préoccupations (les préoccupations métier d'une part et les préoccupations techniques ou transverses d'autre part).
- L'implémentation de chaque préoccupation (les préoccupations transverses sont regroupées dans des aspects).
- L'intégration du système final.

Cette nouvelle technique de programmation introduit de nouveaux concepts (constructions) [Kiczales 01], qui permettent au développeur de bien implémenter les problématiques. Ce sont :

Le point de jointure (joint point) : endroit précis dans l'exécution d'un programme. Les langages de programmation comme AspectJ permettent de définir les points de jointure suivants :

- L'appel d'une méthode ou d'un constructeur.
- L'exécution d'une méthode ou d'un constructeur.
- L'exécution d'un bloc « catch » qui traite une exception Java.
- L'accès en lecture ou écriture d'un champ.

Le point de coupure (pointcut) : c'est un moyen de spécifier un ensemble de points de jointure. Lorsqu'un point de coupure est atteint, il faut préciser les actions à faire et à quels moments les exécuter. Le tableau I ci-dessous illustre les différents pointcut qu'on peut rencontrer.

Tableau I Différents types de point de coupure.

PointCuts	Description
call (void MaClasse.maMethode(..))	Appel de maMethode de MaClasse prenant un nombre quelconque d'arguments, avec comme type de retour void et n'importe quel droit d'accès (public, private, etc…)
call(* MaClasse.maMethode(..))	Appel de maMethode () de MaClasse prenant un nombre quelconque d'arguments, avec n'importe quel type de retour.
call(MaClasse.new(String))	Appel du constructeur de MaClasse avec un argument de type String
execution (void MaClasse.maMethode())	Exécution de maMethode() de MaClasse retournant void.
set (int MaClasse.x)	Écriture dans le champ x de type entier (int) de la classe Maclasse
get(String Toto.nom)	Lecture du champ nom de type String de la classe Toto
Handler(IOException)	Exécution d'un bloc "catch" (en java) qui traite une exception de type IOException.

Les advices : ce sont des fragments de code prêts à être greffés au niveau des points de jointure. Ils implémentent une préoccupation transverse. Pour AspectJ, on distingue trois types d'advice comme illustré dans le tableau II.

Tableau II Les types d'advice.

Type d'advice	Utilisation
Before advice	S'exécute avant un point de jointure ou un point de coupure
After advice	S'exécute après un point de jointure ou un point de coupure
Around advice	Remplace complètement un point de jointure ou un point de coupure

II-4. Aspect mining

L'aspect mining est l'ensemble des techniques permettant d'identifier les préoccupations transverses dans le code. Ces techniques, très souvent supportées par des outils, varient suivant les symptômes de préoccupation à identifier et de l'analyse à exécuter. On distingue deux approches d'identification des préoccupations suivant le type de symptôme qu'elles identifient: l'approche basée sur la duplication de code et celle basée sur sa dispersion.

Les Symptômes de type duplication du code : Les premiers travaux dans cette classe d'approches ont été effectués par J. Yuan et al. [J. Yuan 99]. Ils ont développé Aspect Browser. C'est un environnement de développement permettant une identification des préoccupations transverses basée sur une analyse du texte (analyse lexicale). Le développeur spécifie un clone qui peut être une expression régulière ou une chaine de caractères qu'il soupçonne être une préoccupation transverse. L'outil analysera le code pour identifier le clone (ou l'expression spécifiée) en tout endroit du code où il se trouve et l'affichera avec la couleur choisie dans l'éditeur de code source.

AMT (ou Aspect mining Tool) développé par Hannemann et Kiczales [Hannemann 01], est une extension de Aspect Browser. Il analyse le code se basant sur le type (type d'un objet) ou le texte (expression particulière). L'utilisation de cet outil, en appliquant une seule des deux techniques, peut donner des faux positifs (des éléments identifiés comme préoccupation transverse alors qu'ils ne le sont pas réellement); ceci arrive aussi lorsque la convention de nommage n'est pas respectée. Il est donc conseillé de les combiner pour optimiser le résultat. Une extension de l'outil est en étude, avec l'ajout d'une nouvelle technique basée sur la signature. Cette nouvelle technique est, cependant, moins flexible que les premières. L'outil utilise le concept de Seesoft (outil de visualisation) pour afficher les résultats. Ainsi, le développeur analyse le code, en définissant des questions ou des critères basés soit sur le type d'objet, soit sur une expression particulière, soit sur les deux. Les préoccupations transverses

correspondantes s'affichent en différentes couleurs suivant les critères retenus. Si plusieurs critères s'appliquent par exemple dans une ligne de code, on aura plusieurs couleurs dans la ligne; chacune identifiant une préoccupation transverse particulière.

Le Prisme, développé par Zhang et *al.* [Zhang 03], est une extension d'AMT. En plus des deux techniques d'AMT, l'outil considère le rang du type d'un objet. En effet, les auteurs partent de l'hypothèse que le type d'un objet le plus utilisé dans un code est un bon signe de préoccupation transverse. Ainsi, le type le plus utilisé est classé au premier rang. Cet outil implémente trois techniques, à savoir: le Prisme aspect fingerprint (input) qui permet de représenter les préoccupations transverses, le Prisme aspect footprint (output) qui représente l'endroit correspondant à la préoccupation, et le Prisme aspect mining algorithme qui est un algorithme qui prend en entrée le fingerprint et donne en sortie le footprint.

Limite **:** L'inconvénient avec ces outils d'analyse lexicale est qu'ils exigent une profonde connaissance du code à analyser; d'abord parce que cette analyse dépend en général des pratiques de codage du programmeur, comme la convention de nommage qui n'est pas toujours respectée. Mais, aussi à cause de l'intervention humaine pour formuler l'input; ce qui n'est pas évident [Bounour 06].

Par ailleurs, d'autres travaux d'identification de préoccupations transverses basée sur la duplication du code ont utilisé la technique de détection du clone sans intervention humaine. Nous pouvons citer, parmi ces travaux, le Graphe de Dépendance de Programme (PDG) de Shepherd [Shepherd 04], qui est un graphe contenant des informations de nature sémantique, comme le flux de contrôle et de données du programme. Aussi, l'approche de Bruntink et *al.* [Bruntink 04], qui est une technique hybride, qui combine la technique de l'Arbre Syntaxique Abstrait (AST) basée sur la détection du clone et un outil d'analyse lexicale du code. L'outil utilise un « parser » qui permet d'obtenir la représentation syntaxique du code source en un arbre syntaxique

abstrait (AST), et l'algorithme de détection de clone se charge de retrouver des sous-arbres similaires dans l'arbre syntaxique abstrait.

Limite : Ces techniques souffrent de quelques limitations. Seules les préoccupations de type homogènes sont identifiées. L'identification des préoccupations intéressantes peut échapper à l'analyse. Par ailleurs, le filtre des préoccupations potentielles n'est pas totalement automatisé [Bounour 06].

Tonella et Ceccato [Tonella 04], dans leur approche d'identification des préoccupations transverses, ont développé l'outil Dynamo. Cet outil supporte la technique d'analyse formelle des concepts (analyse dynamique du code). Le concept est un ensemble constitué d'objets et des propriétés (attributs) que ces objets ont en commun. Dans leur étude de cas, ils ont définit comme objets des cas d'utilisation associés aux fonctions principales du programme à analyser, et les attributs sont les méthodes invoquées chaque fois que les cas d'utilisations sont exécutés. Le concept résultant de cette relation est un aspect candidat (préoccupation transverse) si l'une au moins des conditions suivantes est respectée:

- Les méthodes du concept appartiennent à plus d'une classe.
- Différentes méthodes d'une même classe figurent dans plus d'un cas d'utilisation.

Nous pouvons observer au tableau III une illustration de l'analyse des concepts formels. Soit une application donnée qui implémente les classes A et B avec les fonctions principales suivantes : insertion des données et recherche pour comparaison de ces données.

Tableau III Relation entre Objets (cas d'utilisation) et les attributs des classes.

Objet\ Attributs	A.A()	A.inserer()	A.rechercher()	B.B()	B.inserer()	B.rechercher()
Insertion	X	X		X	X	
Recherche	X		X			X

De cette relation résulte l'arbre des concepts suivant.

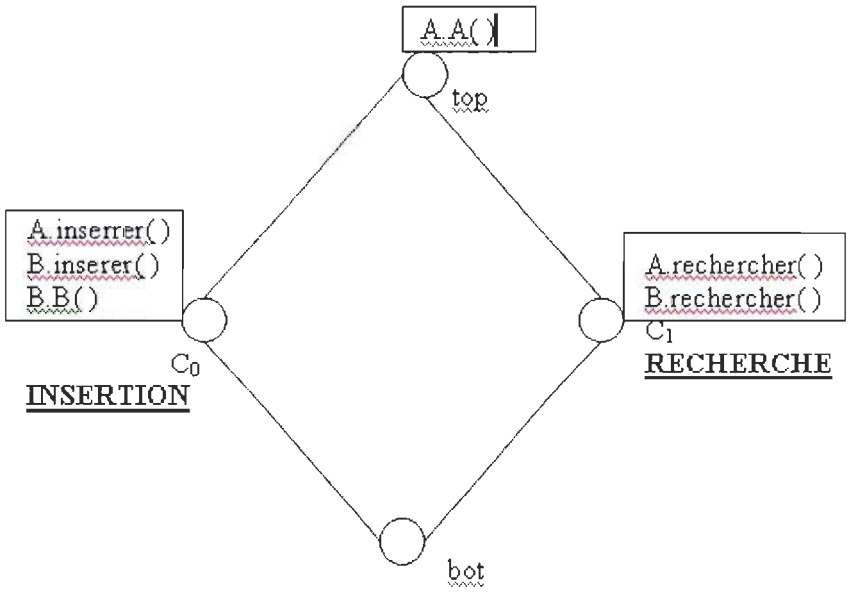


Figure 2: Arbre des concepts.

Les concepts résultants de cet arbre sont :

bot = ({}, {A.A( ), A.**inserer**( ), A.**rechercher**( ), B.B( ), B.**inserer**(), B.**rechercher**( )})
C_0 = ({Insertion}, {A.A( ), A.**inserer**( ), B.B( ), B.**inserer**( )})
C_1 = ({Recherche}, {A.A( ), A.**rechercher**( ), B.**rechercher**( )})
top= ({Insertion, Recherche}, {A.A( )})

Nous remarquons que les méthodes « insérer » et «rechercher » appartiennent à plus d'une classe et apparaissent à la fois dans les cas d'utilisation « insertion » et « recherche ». D'après Tonella et al., les concepts C_0 et C_1 sont des candidats aspects comme illustré dans la figure 2.

D'autres chercheurs comme Tourwé et *al.* [Tourwé 04] ont proposé une approche similaire, mais admettent qu'on retrouve le reflet des préoccupations transverses dans le code en respectant la convention de nommage des classes et méthodes dans le système.

Limite : La difficulté dans ces approches réside dans le fait de décider manuellement si le concept identifie un aspect ou non.

Les symptômes de type dispersion du code : Plusieurs études se sont intéressées à cette catégorie de préoccupations transverses. La technique d'analyse fan-in (analyse statique du code) de Marin et *al.* [Marin 04] est un plug-in Éclipse IDE. L'outil Fint [Marin 04] supporte cette technique. La technique analyse le code pour identifier les préoccupations qui se présentent sous forme d'appel de méthodes. Le fan-in d'une méthode donnée est une métrique qui calcule le nombre de fois que cette méthode apparaît dans le corps d'une autre méthode. L'analyse du code d'un programme par cette technique consiste en trois étapes :

- Calcul automatique de la métrique fan-in de chacune des méthodes du système analysé; le résultat s'affiche sous la forme d'un arbre dont les nœuds sont les méthodes appelées et les branches les méthodes appelantes.

- Le filtre des résultats, exclu toutes les méthodes de fan-in inférieur au seuil choisi, les constructeurs, les accesseurs, les méthodes utilitaires comme toString.
- L'analyse manuelle du résultat du filtre (les méthodes appelées). Ce dernier résultat est l'ensemble des candidats aspects.

La première étude portant sur l'analyse dynamique du code Java pour identifier les symptômes de préoccupations transverses de type dispersion est celle de Breu et *al.* [Breu 03]. Ils ont réalisé la technique d'analyse des patterns récurrents des traces d'exécution. Leur objectif était de retrouver dans les traces d'exécution d'un programme, les patterns récurrents comme un appel d'une méthode M donnée suivi par celui de la méthode N (outside aspect); ou un appel de la méthode M qui apparaît à l'intérieur de l'appel de la méthode N (inside aspect). Ces deux classes se divisent chacune en deux autres sous-classes, soient before ou after pour la première et first-in ou last-in pour la deuxième.

La figure 3 présente une application de la technique des patterns récurrents sur une trace de programme; de cette analyse, il ressort que :

C.d( ) est appelée 4 fois immédiatement à l'intérieur de B.a( ).
G.h( ) est appelée 2 fois avant H.g( ) .
B.a( ) est appelée 2 fois après A.b( ) .
H.i( ) est appelée 2 fois à la fin de F.f( ) .

D'après les conclusions de Breu et *al.*, ces différents patterns permettent de localiser les préoccupations transverses et les méthodes qui y figurent sont des aspects candidats à la « modularisation ».

```
1  public class Trace
2  {B.a(){
3      C.d(){
4          G.h(){}
5          H.g(){}
6      }
7  }
8  A.b(){}
9  B.a(){
10     C.d(){}
11 }
12 A.b(){}
13 B.a(
14     C.d(){
15         G.h(){}
16         H.g(){}
17     }
18     C.c(){}
19 }
20 F.f(){
21     H.i(){}
22 }
23 C.c(){}
24 G.h(){}
25 H.g(){}
26 A.b(){}
27 B.a(){
28     C.d(){}
29     F.f(){
30         H.k(){}
31         H.i(){}
32     }
33 }
34
```

Résultat de l'analyse

Inside aspect :

C.d( ) est appelée 4 fois « first-in » à l'intérieur de B.a( )

H.i( ) est appelée 2 fois « last-in » à l'intérieur de F.f( )

Outside aspect :

G.h( ) est appelée 2 fois « before » H.g( )

B.a( ) est appelée 2 fois « after » A.b( )

Figure 3: Trace de programme à analyser par la technique des patterns récurrents.

D'autres études ont développé des outils permettant de naviguer plus intelligemment dans le code source d'un programme. C'est le cas de Feat [Robillard 02] et JQuery [Volder 02]. Ces outils possèdent des informations de type sémantique et permettent à l'utilisateur d'avoir le contrôle de son analyse par les fonctions qu'ils lui offrent.

Limite : L'utilisation de ces outils requiert une connaissance considérable de la structure et des fonctions du programme à analyser; et nécessite aussi beaucoup de temps pour identifier un aspect candidat.

Dans le tableau IV, nous récapitulons les différentes approches, leurs techniques et les outils qui les soutiennent.

Tableau IV Classification des différentes approches et les outils qui supportent leurs techniques.

Approche	Technique	Outil	Type d'analyse	Résultat
Duplication du code	Détection du clone	Aspect browser, AMT, Prism	Lexical, type, rang du type	concept de Seesoft
		Ophir	Détection de clone – PDG (graphe de dépendance)	Liste d'aspects candidats manuellement détecté
	Analyse de concepts formels (FCA	Delfstof	Analyse des traces d'exécution (objets/attribut)	idem
		Dynamo	Analyse des traces d'exécution	idem
Dispersion du code	Analyse dynamique	Dynamit	Patterns récurrents	Trace d'un programme
	Analyse de renvoi	Fint	Appel des méthodes	Arbre des appels de méthodes
Explorateur	Exploratoire	JQuery. Feat, Sextant	Analyse sémantique (flux de contrôle)	Explorateur intelligent

CHAPITRE III

COHÉSION DANS LES SYSTÈMES ORIENTÉS OBJET

III-1.Métriques Orientées Objet

Le développement ainsi que la large utilisation des technologies orientées objet ont apporté une contribution importante au développement des systèmes logiciels. Ces technologies ont essentiellement contribué à l'amélioration de la qualité des logiciels développés. La qualité des logiciels est définie à travers plusieurs caractéristiques importantes telles que la fiabilité, la maintenabilité et la réutilisabilité. Plusieurs attributs logiciels (tel que la complexité) contribuent (directement ou indirectement) à la définition de ces caractéristiques.

La fiabilité [Sommerville 04] définit la probabilité qu'au cours d'une période donnée le système puisse répondre correctement aux attentes de l'utilisateur. La complexité, par contre, dépend grandement des habitudes et de la méthode du développeur. Certaines méthodes se donnent pour objectif de la réduire, ce qui augmente la qualité [K. Liu 00]. Mesurer ces facteurs revient à mesurer certains attributs de qualité qui les caractérisent comme la taille du système, sa complexité, le couplage, la cohésion, l'héritage, l'encapsulation, le polymorphisme et la réutilisation.

Les métriques relatives à la taille d'un système nous renseignent, par exemple, sur le nombre de méthodes et d'attributs dans une classe du système [K. Aggarwal 06], ce qui intervient dans le niveau de complexité du système. Les métriques telles que NOA (nombre d'attributs par classe), NOM (nombre de méthodes par classe) [Henderson 96], WMC (la complexité totale des méthodes), et RFC (ensemble des méthodes

potentiellement exécutées en réponse au message reçu par un objet de la classe [Chidamber 94]) permettent d'évaluer cet attribut.

Le couplage, un autre attribut de qualité, mesure le degré auquel un élément est lié à un autre [Larman 05]. Un fort couplage augmente la complexité, réduit la réutilisabilité, et rend difficile la compréhension ainsi que la maintenance. Parmi les métriques qui évaluent le couplage entre classes, nous pouvons citer: CBO (couplage ente objets) [Chidamber 94], DAC (couplage par type de données abstraites), et MPC (couplage par passage de message) [Henderson 96].

L'héritage, l'encapsulation et le polymorphisme sont aussi des éléments qui ont un impact sur la qualité des programmes. Les métriques DIT (la profondeur de l'héritage) et NOC (nombre de sous-classes d'une classe), deux métriques de la suite des métriques de Chidamber et Kemerer [Chidamber 94], renseignent sur l'héritage.

Par ailleurs, la cohésion des classes est considérée comme l'un des attributs les plus importants des systèmes orientés objet. Elle réfère au degré de liaison des éléments d'une classe. Une faible cohésion (manque de cohésion) est un signe de mauvaise conception. Une classe présentant une faible cohésion est difficile à comprendre, à tester, à réutiliser et à maintenir. Plusieurs métriques de cohésion orientées objet ont été proposées dans la littérature: LCOM [Chidamber 94], TCC et LCC [Bieman 95], DCi et DCd [Badri 04], DCie et DCde [Badri 08]. La métrique LCOM à connu plusieurs variantes.

Dans ce qui suit, nous présentons les métriques de cohésion utilisées dans notre étude.

III-2. Métriques de cohésion

La cohésion mesure le degré de liaison entre les parties d'un composant logiciel qui s'investissent pour effectuer une tâche bien définie [Larman 05]. Elle décrit la façon

dont les composants sont liés. Une forte cohésion est fortement souhaitée en conception. Elle facilite les modifications et augmente la qualité [Sommerville 04].

III-2.1. Approche de Chidamber et Kemerer

LCOM 1: Lack of Cohesion in Methods

La première métrique pour évaluer la cohésion des classes LCOM (Lack of COhesion in Methods) a été proposée par Chidamber et Kemerer [Chidamber 91]. Elle définit la relation directe entre les méthodes et les variables d'une classe. Plus tard, elle va subir des variations. Dans un premier temps, les auteurs considèrent une classe C, avec {Mi, 1<i<n} l'ensemble formé de ses méthodes, et {Ii, 1<i<n} l'ensemble des instances de variables utilisées par chacune des méthodes Mi. LCOM est donnée par le nombre de paires de méthodes disjointes (ne partageant aucune variable d'instance en commun). Plus le nombre de paires de méthodes ne partageant aucune variable d'instance en commun est grand, plus le manque de cohésion est important. Cela renseigne, en quelque sorte, la disparité dans les méthodes de la classe.

Cette définition de la cohésion a fait l'objet de plusieurs critiques dans la littérature et a connue plusieurs améliorations dont celle de ses propres auteurs LCOM1 [Chidamber 94]. LCOM1 a été définie comme étant le nombre de paires de méthodes ne partageant pas d'instances de variables moins le nombre de paires de méthodes partageant des instances de variables de la classe. Si cette différence est négative, alors LCOM1 est fixée à zéro.

Les limites de cette définition se font sentir lorsqu'une classe dont le nombre de paires de méthodes ne partageant aucune instance de variable est égal au nombre de paires de méthodes partageant au moins une instance de variable. LCOM sera, en fait, nulle sans que la classe soit forcément cohésive. Il en est de même pour les classes de cohésion

nulle dont certaines sont plus cohésives que d'autres. Plusieurs travaux ont été réalisés critiquant l'approche de Chidamber et *al.*, dont ceux de Li et *al.* [Li 95].

III-2.2. Approche de Li et Henry

Redéfinition de LCOM

Li et *al.* [Li 95] partent de l'hypothèse que si toutes les méthodes définies dans une classe accèdent à des ensembles différents de données encapsulées dans la classe, la classe serait mal conçue. Ils proposent alors de définir (redéfinir en fait) LCOM comme le nombre d'ensembles disjoints de méthodes locales d'une classe [Li 95]. On entend ici par ensemble disjoint, un groupe d'ensembles de méthodes qui n'ont aucune méthode commune; et chaque ensemble ne contient que les méthodes qui partagent au moins une variable locale.

III-2.3. Approche de Hitz et Montazeri

Redéfinition graphique de l'approche de Li et *al.*

Hitz et *al.* [Hitz 95] redéfinissent l'approche de Li et *al.* en appliquant la théorie des graphes. Ils définissent LCOM comme le nombre de composantes connexes du graphe. Les sommets de ce graphe représentent les méthodes de la classe. Il y a connexion entre deux sommets, si les méthodes correspondantes accèdent à une même variable d'instance. Ils étendent cette approche en considérant un critère de cohésion supplémentaire relié aux appels de méthodes. Ainsi, selon ce critère, deux méthodes Mi et Mj sont connectées (reliées) si Mi fait appel à Mj et vice versa.

III-2.4. Approches de Brian Henderson-Sellers

LCOM2 et LCOM3

La proposition de Chidamber et Kemerer de la cohésion LCOM1 donne des valeurs nulles pour différentes classes. Pour résoudre ce problème, Henderson-Sellers [Henderson 96] propose LCOM2 et LCOM3 qui sont des métriques similaires avec des formulations différentes. Ces métriques sont implémentées (et calculées) par l'outil Together (Borland) [Borland 08].

Cette approche considère que pour une classe C, avec m le nombre de ses méthodes et a le nombre de ses variables : m(a) définit le nombre de méthodes qui accèdent à une variable de la classe, et Sum(m(a)) est la somme des m(a) pour l'ensemble des variables de la classe C. D'après Henderson-Sellers :

LCOM2 = 1- sum (m(a))/m*a

Si m =0 ou a =0 (classe abstraite, ou sans variable), LCOM2 n'est pas définie et donc fixée à zéro.

LCOM2 est définie comme le pourcentage de méthodes qui n'accèdent à aucune variable dans l'ensemble des variables de la classe.

LCOM3 = [m – sum (m(a))/a]/(m-1)]

Si m =1 ou a =0 (classe ayant une seule méthode; ou classe sans variable), LCOM3 n'est pas définie et donc fixée à zéro.

La figure 4 nous donne une application du critère de partage de la variable entre les méthodes d'une même classe.

Une faible valeur de LCOM est fortement recommandée. Elle implique une forte cohésion, donc une bonne conception. Ceci qui augmente la compréhension et la réutilisation de la classe. Une classe cohésive présente une bonne encapsulation. Une valeur forte de LCOM (manque de cohésion) augmente la complexité et la probabilité d'erreur.

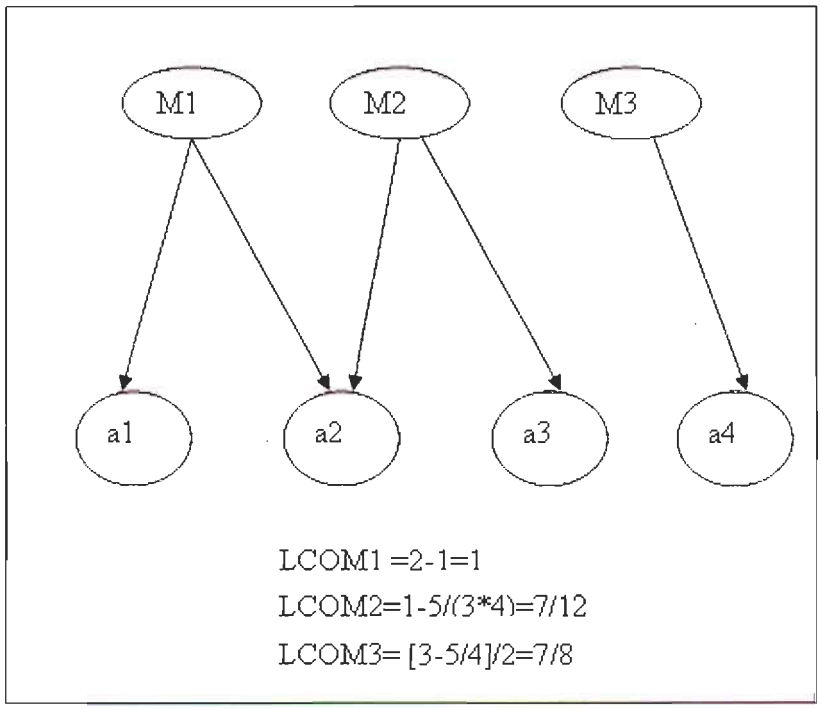


Figure 4 : Connexions entre les éléments d'une classe (partage des variables).

III-2.5. Approche de Bieman et *al.*

TCC et LCC

La proposition de Bieman et *al.* [Bieman 95] pour la mesure de la cohésion, TCC (Tight Class Cohesion) et LCC (Loose Class Cohesion), est étroitement liée à celle de Hitz et *al.*, même s'il y a quelques différences. Ils considèrent le critère de partage de variables d'instances et d'appel de méthodes (figure 5). Une variable d'instance peut être utilisée directement ou indirectement par une méthode. Une variable d'instance est directement utilisée par une méthode Mi si elle apparaît dans le corps de cette méthode. Elle est indirectement utilisée par une méthode Mi, si elle est directement utilisée par une méthode Mj qui elle même est directement ou indirectement appelée par Mi. Deux méthodes sont directement liées ou connectées, si elles utilisent en commun directement ou indirectement au moins une variable d'instance. TCC est définit comme le pourcentage de paires de méthodes publiques de la classe qui sont directement connectées. Dans une classe de N méthodes, le nombre de connexions possibles est donné par la formule : **NC= N*(N-1)/2 (1).**

Alors, on a : **TCC= NCd/NC (2),** où NCd est le nombre de connexions directes entre les méthodes.

En plus des connexions directes entre les méthodes d'une classe, LCC prend en compte les connexions indirectes qui peuvent exister entre elles. LCC est définie comme le pourcentage de paires de méthodes publiques d'une classe qui sont directement ou indirectement connectées. La relation indirecte correspond, en fait, à la fermeture transitive de la relation directe. LCC est donnée par :

LCC = (NCd + NCid)/NC (3), où NCid désigne le nombre de connexions indirectes entre les méthodes.

Les valeurs données par les métriques TCC et LCC sont comprises entre 0 et 1. La valeur 0 correspond à une classe qui ne présente pas de cohésion du tout. La valeur 1 correspond à une classe dont la cohésion est parfaite.

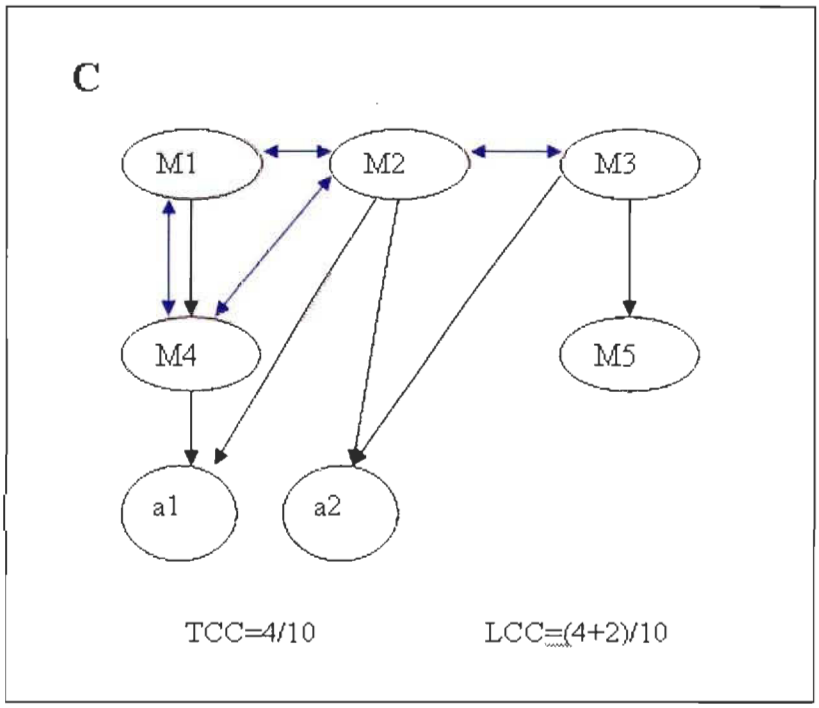


Figure 5: Connexions entre les éléments d'une classe (Partage de variables, appels de méthodes).

III-2.6. Approche de Badri et *al.*

DCd et DCi

Cette approche [Badri 04], en plus de la connexion entre les méthodes publiques d'une classe par partage de variables, considère aussi l'interaction entre les méthodes de la classe. Elle étend le critère d'appel de méthodes [Badri 95] d'une part, introduit le concept de l'utilisation indirecte (la transitivité de la relation directe) de variables d'instances comme définie dans [Bieman 95] et l'étend au critère d'appel de méthodes d'autre part. Deux méthodes Mi et Mj d'une classe C sont directement reliées si :

(i) elles ont en commun au moins une variable,
(ii) ou elles interagissent au moins avec une même méthode de la classe,
(iii) ou elles interagissent entre elles.

La métrique DCd représente le degré de cohésion d'une classe donnée, basée sur la relation directe entre ses méthodes publiques. Elle se définit comme le pourcentage de paires de méthodes publiques qui sont directement reliées. En appliquant la théorie des graphes, les auteurs ont considéré un graphe orienté G dont les sommets représentent les méthodes de la classe, et |Ed| le nombre de connexions directes dans ce graphe. Sachant que le nombre maximum de paires de méthodes dans une classe de N méthodes est donné par N*(N-1)/2, alors DCd=|Ed|/ N*(N-1)/2 (4) (indique le degré de cohésion). Dans ce cas également, la valeur de DCd est comprise entre 0 et 1 (même interprétation que TCC).

De même, deux méthodes sont indirectement liées si elles sont directement ou indirectement (transitivité de la relation directe) liées à une autre méthode.

La métrique DCi représente le degré de cohésion d'une classe donnée, basée sur les relations directes et indirectes entre ses méthodes publiques. Elle se définit comme le pourcentage de paires de méthodes publiques qui sont directement ou indirectement liées. La métrique DCi est donnée par : DCi=|Ei|/ N*(N-1)/2 (5), où |Ei| désigne le nombre de connexions directes et indirectes dans le graphe. Dans ce cas également, la valeur de DCi est comprise entre 0 et 1 (même interprétation que LCC).

DCde et DCie

En plus du critère de partage de variables d'une classe, du critère d'interaction entre les méthodes d'une classe, cette autre approche de Badri et *al.* [Badri 08] considère le partage de paramètres de type objet entre les méthodes d'une classe. Cette extension a été proposée suite à plusieurs expérimentations effectuées en utilisant les premières métriques. Ainsi, deux méthodes publiques sont directement liées si elles remplissent au moins une des conditions suivantes :

(i) elles ont en commun au moins une variable,
(ii) elles ont en commun au moins un objet passé en paramètre,
(iii) elles interagissent entre elles ou en commun avec au moins une autre méthode de la classe.

Le degré de cohésion basé sur la relation directe ainsi définie est donné par DCde qui est une extension de DCd, et représente le pourcentage de paires de méthodes publiques d'une classe directement reliées. Le degré de cohésion basée sur les relations directes et indirectes (transitivité de la relation directe) donné par DCie, qui est une extension de DCi, représente le pourcentage de paires de méthodes publiques d'une classe directement ou indirectement liées. Ces deux métriques, comme les précédentes, sont aussi normalisées.

Cette approche de la cohésion capture plus d'information (quant aux paires de méthodes effectivement reliées) que les autres approches de cohésion, particulièrement celles qui ont utilisé le critère d'interaction entre les méthodes.

Pour les besoins de notre étude, de toutes ces métriques, nous avons retenu les approches de LCOM calculées par l'outil Borland Together et les approches basées sur les métriques DCd, DCi, DCde, DCie, TCC et LCC. Nous reviendrons sur certains détails liés à ces métriques (variantes implémentées et utilisées dans nos expérimentations) plus loin dans le mémoire.

CHAPITRE IV

COHÉSION COMME INDICATEUR DE PRÉSENCE DE PRÉOCCUPATIONS TRANSVERSES

IV-1. Définition de l'approche

La « modularisation » des éléments transversaux dans les systèmes orientés objet est un processus organisé en plusieurs étapes. Parmi ces étapes, figure celle portant sur l'identification et la localisation de ces éléments. Elle consiste essentiellement à retrouver ces « bouts de code » qui sont dupliqués dans plusieurs endroits du programme. Notre approche s'inscrit dans ce cadre. Notre objectif étant d'explorer, comme nouvelle piste de recherche dans le domaine de l'aspect mining, l'utilisation de la cohésion pour supporter l'identification des classes contenant des préoccupations transverses. En fait, notre approche consiste à utiliser la cohésion comme un indicateur de disparité du code, ce qui pourrait être considéré comme un symptôme de code contenant des préoccupations transverses. L'objectif est donc d'évaluer jusqu'à quel degré la cohésion des classes pourrait être utilisée pour déceler par analyse statique automatique de code la présence de préoccupations transverses. Si jamais cette hypothèse est validée, la cohésion pourrait être utilisée pour orienter des actions de « refactoring aspect ».

IV-2. Méthodologie de notre approche

Le recoupement des préoccupations dans un code augmente sa complexité et réduit sa réutilisabilité. Il affecte, à plusieurs niveaux, sa qualité. Pour arriver à expérimenter notre approche et évaluer sa pertinence, nous avons adopté une démarche organisée en plusieurs étapes. Notre méthodologie consiste à :

- ✓ Identifier les symptômes de dispersion de code (classes candidates) en utilisant un outil d'aspect mining validé: nous analysons un code écrit en Java à l'aide de la technique Fan-in de l'outil Fint (un plug-in Éclipse), outil validé dans le domaine de l'aspect mining, pour capturer les classes Ci (classes candidates) présentant ce qui semble être les symptômes de code correspondant à des préoccupations transverses comme présenté dans la figure 6.

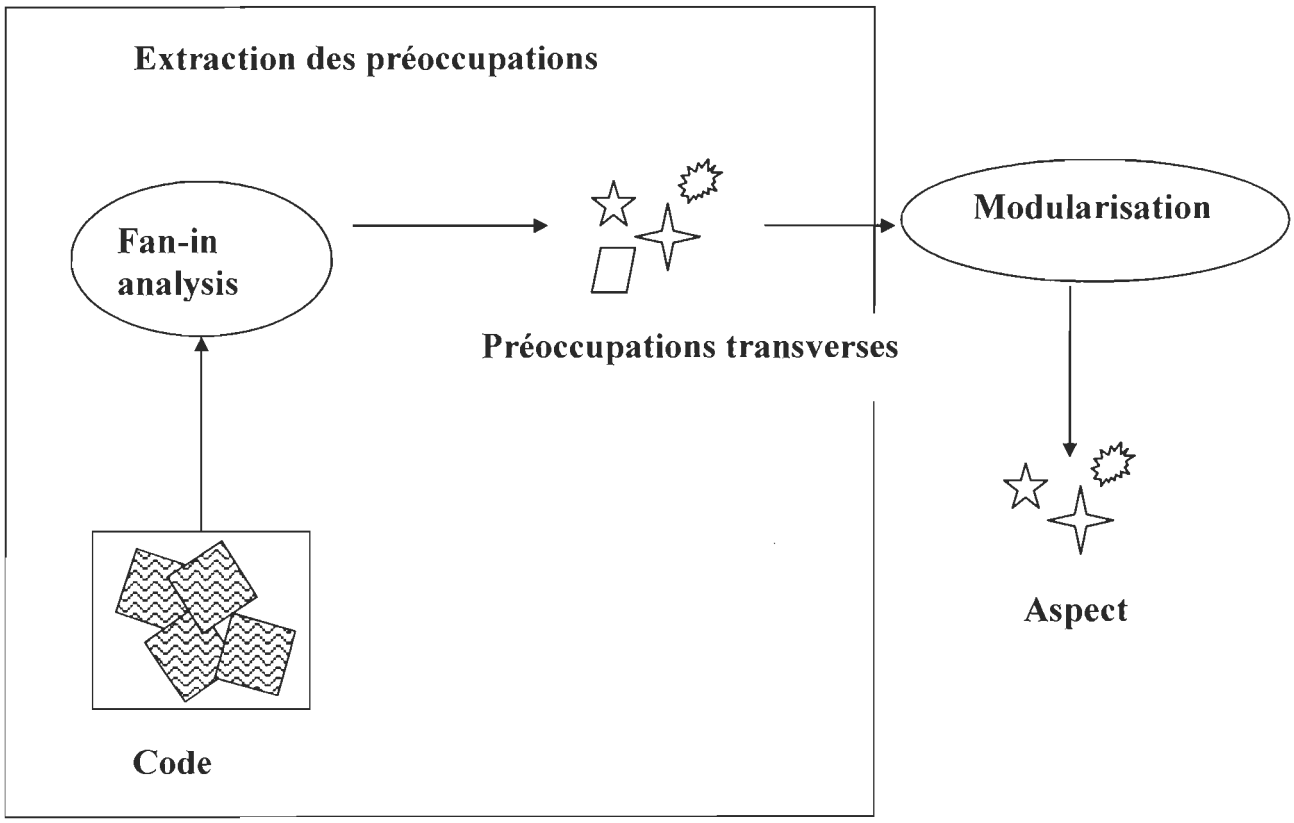


Figure 6:Extraction des fonctionnalités transversales par la technique fan-in.

- ✓ Sélectionner et évaluer les métriques de cohésion retenues: nous avons sélectionné plusieurs métriques de cohésion suivant leurs critères d'évaluation de

la cohésion des classes. Certaines évaluent la cohésion uniquement suivant le critère de partage de variables; d'autres, en plus du critère de partage de variables, utilisent aussi le critère d'appel de méthodes. Puis, un dernier groupe de métriques qui, en plus des deux critères mentionnés, prend en compte le critère de partage d'objets passés en paramètre. Nous évaluons par la suite la cohésion pour chacune des classes C_i de chacun des projets sélectionnés.

- ✓ Nous faisons la collecte, l'analyse et l'interprétation des données.

- ✓ Visualiser les tendances des résultats: avant d'utiliser des méthodes d'analyse de données avancées comme la régression logistique, il est nécessaire dans un premier temps, de découvrir les données afin d'identifier des tendances, de repérer des anomalies ou tout simplement de disposer d'informations essentielles telles que le minimum, le maximum, ou la moyenne d'un échantillon de données.

- ✓ Évaluer les corrélations entre la cohésion et les classes candidates: nous effectuons une étude de corrélation entre les classes C_i contenant réellement du code correspondant à des préoccupations transverses (identifiées par l'outil Fint d'aspect mining) et leurs métriques de cohésion pour vérifier nos hypothèses.
  Cette démarche se résume schématiquement dans la figure 7.

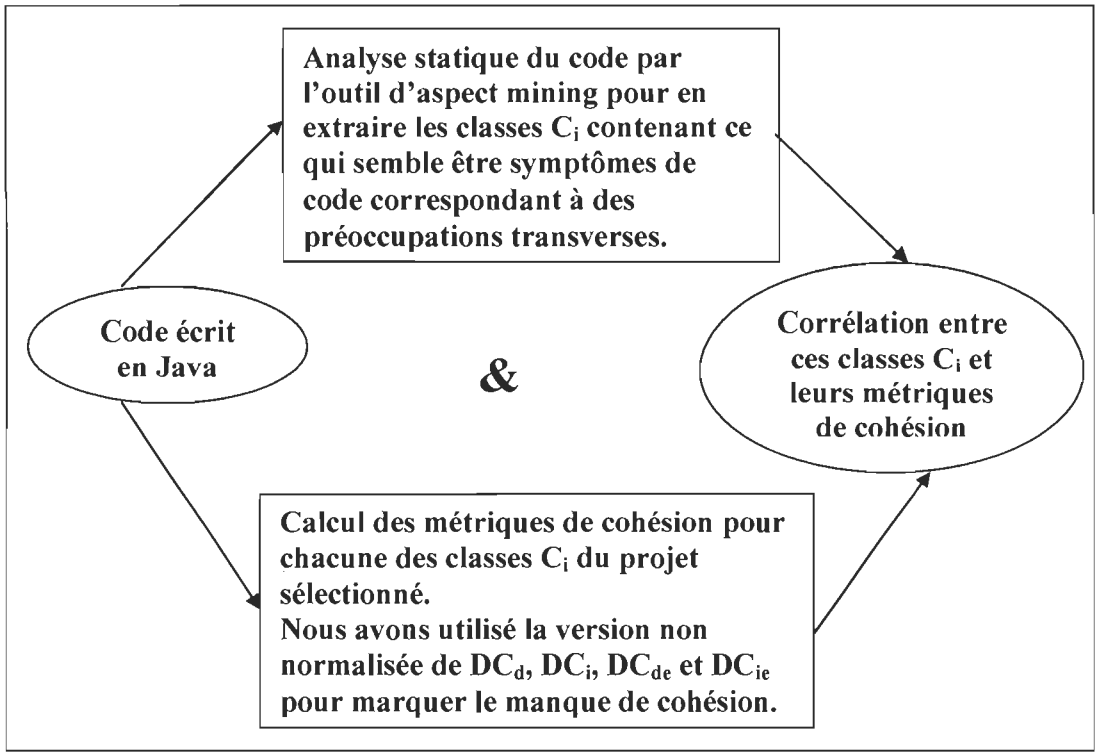


Figure 7: La représentation schématique de notre démarche.

IV-3. Environnement de travail

Notre environnement de travail est Éclipse (3.4) [Éclipse 08]. C'est un environnement de développement intégré, libre, extensible, universel et polyvalent. Éclipse permet potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Éclipse IDE (Integrated Development Environment) est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM). Ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions. Éclipse utilise énormément le concept de "plug-in" dans son architecture. D'ailleurs, l'ensemble de sa plate-forme est développé sous forme de plug-ins excepté le noyau appelé "Runtime". Ce concept permet de fournir un mécanisme pour l'extension de la plate-forme et ainsi fournir la possibilité à des tiers de développer des fonctionnalités qui ne sont pas fournies en standard par Éclipse.

Les principaux modules fournis en standard avec Éclipse concernent Java. Des modules sont, cependant, en cours de développement pour d'autres langages, notamment C++, Cobol, AspectJ, et aussi pour d'autres aspects du développement (base de données, conception avec UML, ...). Ils sont tous développés en Java, soit par le projet Éclipse soit par des tiers commerciaux ou en open source. Les modules agissent sur des fichiers qui sont inclus dans l'espace de travail ou Workspace. L'espace de travail regroupe les projets qui contiennent une arborescence de fichiers. Bien que développé en Java, les performances à l'exécution d'Éclipse sont très bonnes, car il n'utilise pas Swing pour l'interface homme-machine mais un toolkit particulier appelé SWT (Standard Widget Toolkit) associé à la bibliothèque JFace. SWT est développé en Java par IBM en utilisant au maximum les composants natifs fournis par le système d'exploitation sous jacent. JFace utilise SWT et propose une API pour faciliter le développement d'interfaces graphiques. Éclipse ne peut donc fonctionner que sur les plate-formes pour lesquelles SWT a été porté. Ainsi, Éclipse 1.0 fonctionne uniquement sur les plates-formes Windows 98/NT/2000/XP et Linux. SWT et JFace sont utilisés par Éclipse pour développer le plan de travail ou « Workbench » qui organise la structure de la plate-forme et les interactions entre les outils et l'utilisateur. Cette structure repose sur trois concepts: la perspective, la vue et l'éditeur. La perspective regroupe des vues et des éditeurs pour offrir une vision particulière des développements. En standard, Éclipse propose huit perspectives. Les vues permettent de visualiser et de sélectionner des éléments. Les éditeurs permettent de visualiser et de modifier le contenu d'un élément de l'espace de travail. L'architecture d'Éclipse étant composée dans l'ensemble de plug-in, l'outil (Fint) d'analyse de code pour identifier les préoccupations transverses utilisé dans ce travail est un plug-in Éclipse. Grâce à cette architecture, nous pouvons exécuter sur Éclipse l'analyse fan-in qui est une des techniques qu'offre cet outil et qui n'est pas disponible dans Éclipse standard.

IV-4. Outil d'aspect mining: FINT

IV-4.1. Description de l'outil

Fint est un outil d'analyse de code source Java permettant d'extraire ce qui semble être du code correspondant à des préoccupations transverses. Le Fint est disponible comme plugin Éclipse IDE. Cet outil supporte trois techniques d'analyse de code et son architecture comprend trois vues correspondant à ces techniques [Marin 08] :

- ✓ Le fan-in analysis view (ou l'analyse fan-in).
- ✓ Le Grouped calls analysis view (analyse d'appels groupés).
- ✓ La redirection finder view (la redirection).

L'analyse Fan-in recherche, en fait, les préoccupations transverses en investiguant tous les appels de méthodes dans le système. Elle permet à l'utilisateur de sélectionner les méthodes qui sont appelées à plusieurs endroits différents, possiblement dans des contextes similaires. Elle attache à chaque méthode identifiée toutes les méthodes qui l'appellent. Plus une méthode est appelée dans le code, plus grande est la valeur de sa métrique fan-in; ce qui l'identifie potentiellement comme préoccupation transverse par l'outil Fint.

Tout comme l'analyse fan-in, le « grouped calls analysis » analyse le code en ciblant tout ce qui est préoccupation transverse. Seulement, au lieu des appels singuliers de méthodes, le grouped calls analysis identifie des ensembles formés d'au moins deux méthodes appelées par un même ensemble d'autres méthodes. Également, plus l'ensemble de méthodes appelantes est grand, plus il est probable que le groupe de méthodes appelées soit identifié comme préoccupation transverse (aspect candidat) par la technique.

La redirection (redirections finder) analyse le code du système pour identifier des classes particulières dont les méthodes sont dans une relation exclusive d'appel un à un avec les méthodes d'autres classes.

Bien qu'il est souvent conseillé d'utiliser la combinaison des techniques pour optimiser les résultats, ces deux dernières techniques de l'outil n'ont pas été prises en compte dans notre expérimentation, car seule, ou en combinaison avec une autre, elles ne nous apportaient pas plus d'information que la première prise seule.

IV-4.2. Fonctionnement de l'outil

L'analyse Fan-in est la technique par défaut de l'outil. C'est celle utilisée dans l'ensemble de nos expérimentations. Pour l'exécuter, l'utilisateur sélectionne un projet dans l'explorateur d'Éclipse, et choisi dans le menu contextuel (clique droit sur le projet sélectionné) l'option Fan-in analysis (figure 8). L'outil analyse le code source du projet sélectionné (ou l'élément sélectionné) et construit un modèle en mémoire qui sera utilisé par la suite par toutes les autres techniques supportées par l'outil (le grouped call analysis, et la redirection finder).

Capture d'écran

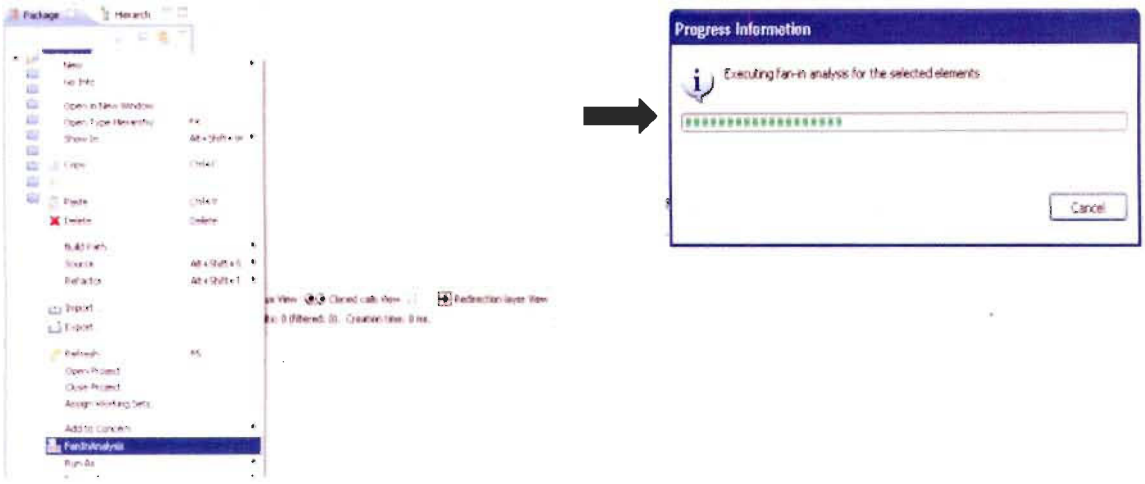


Figure 8: Exécution de l'analyse fan-in.

Le modèle construit en exécutant le Fan-in analysis peut prendre une trentaine de secondes pour un système qui compte 20 000 lignes de code (LOC) et cinq minutes pour un système de 360 000 LOC sur un Pentium 4 (2.66 GHz).

L'analyse fan-in recherche tous les appels de méthodes du système et les affiche sous forme d'arbre dont la racine est la méthode appelée, et les branches les méthodes qui l'appellent (figure 9). La valeur du fan-in de chaque méthode est le nombre de méthodes qui l'appellent. Le fan-in analysis a des options de filtrage pour affiner le résultat. C'est ainsi qu'on peut exclure les méthodes des librairies, les accesseurs; afficher les résultats par nom de méthode ou par valeur de fan-in et aussi filtrer par rapport au seuil de fan-in choisit dans l'expérimentation. Ainsi, toutes les méthodes de fan-in en deçà du seuil sont exclues.

Capture d'écran

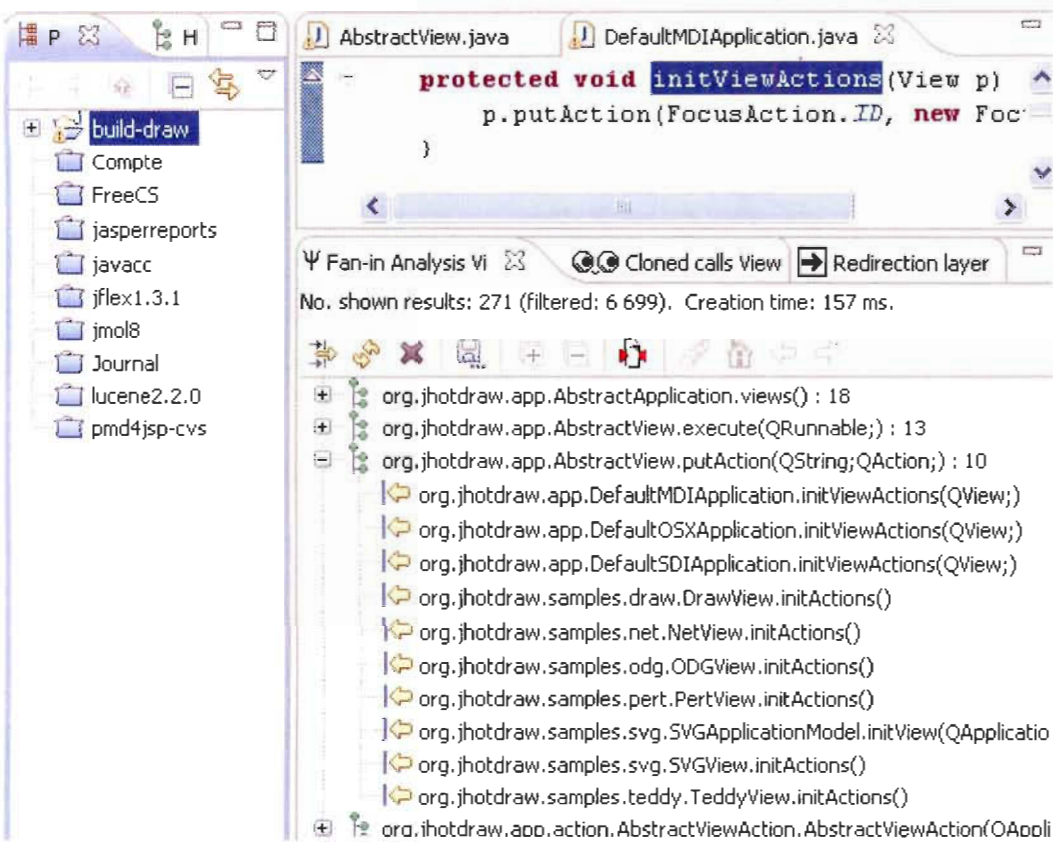


Figure 9: Illustration de l'analyse statique du code par l'outil Fint.

CHAPITRE V

ÉTUDE EMPIRIQUE

V-1. Démarche

Nous avons suivi une démarche organisée en plusieurs étapes :

1- Choix des systèmes à étudier.

2- Analyse statique du code de ces systèmes pour extraire les classes candidates (contenant du code pouvant correspondre à des préoccupations transverses), par la technique Fan-in analysis de l'outil d'aspect mining FINT.

3- Analyse de l'ensemble des résultats donnés par l'outil (recherche des classes candidates à partir de leur valeur de fan-in). La variable réponse "Classe" est une variable binaire. Elle prend la valeur 1 si la classe est considérée comme candidate (classe ayant un fan-in supérieur au seuil) et 0 sinon.

4- Calcul des métriques de cohésion retenues (utilisation de l'outil Together (Borland) ainsi que celui développé par Badri et *al.* [Badri 04, Badri 08]). Nous nous intéressons dans notre étude à la cohésion, et plus exactement au manque de cohésion, ce qui indiquerait une disparité dans le code. Comme les métriques utilisées (LCOM et quelques unes de ses variantes) sont des métriques inverses de la cohésion (indiquant le manque de cohésion), et pour faciliter la comparaison avec les métriques proposées par Badri et *al.* dans [Badri 04, Badri 08], nous utiliserons dans nos expérimentations aussi bien les métriques de cohésion originales proposées par Badri et *al.*, ainsi que des métriques indiquant le manque de cohésion (selon l'approche de LCOM) dérivées des métriques originales.

5- Tracé des diagrammes en boite (pour visualiser les tendances) pour chaque métrique.
6- Étude de la régression logistique (confirmation des tendances observées précédemment) pour chaque métrique.
7- Pour chacun des projets sélectionnés, on répètera les points 2, 3, 4, 5 et 6.

V-2. Environnement et collecte des données

V-2.1. Systèmes analysés

Pour notre étude expérimentale, nous avons évalué huit projets open source Java tirés de plusieurs domaines.

FreeCS v 1.2.2 (http://freecs.sourceforge.net/): FreeCS est un serveur de discussion en ligne (chat server) écrit en Java. Il supporte une interface entièrement personnalisable aussi bien du côté client que du côté serveur.

JMOL8 (http://jmol.sourceforge.net/): JMOL8 est un logiciel open source gratuit de visualisation de molécules, pour les étudiants, les éducateurs et les chercheurs en chimie et biochimie. Il est multiplateforme, fonctionnant sous Windows, Mac OS X et les systèmes Linux/Unix. JMOL a commencé en tant que projet OpenScience (http:// www.openscience.org) dédié à l'écriture et à la livraison de logiciels scientifiques gratuits et open source.

JFlex v 1.3.1 (http://jflex.de/): JFlex est un analyseur lexical et générateur de code source Java. Il reçoit en entrée des spécifications à partir d'un fichier, dans un format spécial, et génère le code source Java de l'analyseur correspondant aux spécifications. Son code source est ouvert au public.

Lucene v 2.2.0 (http://www.apache.org/): Lucene est une librairie complète qui fournit des recherches textuelles. Ses développements sont supportés par « Apache Software Foundation».

EJB (http://jmol.sourceforge.net/): EJB est l'une des principales technologies de la plate-forme J2EE. Elle propose un standard visant à développer des composants Java réutilisables, s'exécutant sur un serveur d'applications.

PMD (http://pmd.sourceforge.net/): PMD est un analyseur de code source Java. Il recherche les variables non utilisées, les objets créés non nécessaires.

JasperReports (http://www.jasperforge.org/): JasperReports est un outil de « reporting », offert sous forme d'une bibliothèque qui peut être embarquée dans tout type d'application Java. Il se base sur des fichiers XML (dont l'extension est en général .jrxml) pour la présentation des états. Il peut être couplé à iReport pour faciliter sa mise en œuvre dans une application Java classique ou orientée web.

JhotDraw (http://jhotdraw.sourceforge.net/): JhotDraw est un Framework (cadre de développement) graphique de dimension deux, écrit en Java pour des schémas structurés.

V-2.2. Métriques de cohésion sélectionnées

Un grand nombre de métriques de cohésion a été proposé dans la littérature pour évaluer la cohésion des classes dans les systèmes orientés objet. Dans le but d'expérimenter notre approche, nous avons sélectionné les métriques suivantes:

LCOM1, LCOM2 et LCOM3 : nous avons utilisé l'outil Togheter (Borland) pour les évaluer.

Ces métriques, comme mentionné précédemment, se basent sur le critère de partage des attributs dans une classe. Plusieurs études ont pu démontrer que ce critère seul était insuffisant pour faire une évaluation efficace de la cohésion. Selon certains auteurs, il ne prend pas en compte certaines caractéristiques de la classe comme la taille, et la connectivité entre ses éléments. Bieman et *al.* [Bieman 95], par exemple, définissent la cohésion dans une classe non seulement par la connexion entre deux méthodes du fait qu'elles partagent au moins une variable d'instance tel que définit par Chidamber et Kemerer [Chidamber 91], mais aussi via un appel de méthode (une variable est utilisée par une méthode si elle est utilisée par la méthode qu'elle appelle).

TCC et **LCC** [Bieman 95] définissent, respectivement, le pourcentage de paires de méthodes publiques de la classe qui sont directement et indirectement connectées.

DCd et **DCi** [Badri 03, Badri 04], en plus du critère de partage des variables d'instances, ces métriques se basent également sur le critère d'invocation de méthodes en commun (même si les méthodes invoquées n'utilisent pas des attributs ou des attributs en commun), de façon directe (DCd) ou indirecte (transitivité de la relation directe - DCi).

DCde et **DCie** [Badri 08] représentent des extensions respectives des deux précédentes métriques. En plus du partage de variables d'instances et de l'invocation de méthodes (de la même manière que précédemment), cette nouvelle approche a expérimenté un troisième critère qui permet de capturer d'autres paires de méthodes connectées que les autres critères ne capturent pas. Il s'agit du critère de partage d'objets passés en paramètre à des méthodes. Nous avons calculé les métriques de Badri et *al.*, en nous servant de l'outil développé basé sur JavaCC. Suivant notre démarche, nous devons donc tester s'il existe une corrélation entre l'absence de cohésion (disparité dans le code) et la présence de fonctionnalités transverses dans un code Java. Pour le bon déroulement de notre expérimentation et pour évaluer nos hypothèses, nous avons expérimenté également la version non normalisée des métriques DCd, DCi, DCde et DCie. Nous

avons, en fait, dérivé de ces métriques des métriques non normalisées indiquant le manque de cohésion (disparité dans le code), ce qui nous a donné les métriques suivantes LDCd, LDCi, LDCde et LDCie. Ces dernières métriques indiquent, en fait, selon l'approche adoptée par les métriques LCOM (pour faciliter la comparaison), le manque de cohésion dans une classe, ce qui correspondrait directement à un indice de disparité dans le code d'une classe. Nous avons procédé de la même manière concernant les métriques TCC et LCC présentées précédemment. Nous avons donc expérimenté des valeurs indiquant le manque de cohésion tirées de ces métriques que nous avons appelées L-TCC et L-LCC.

Nous avons, en fait, au début de nos expérimentations utilisé les métriques normalisées telles qu'elles sont définies par leurs auteurs (DCd et DCi [Badri 04]; DCde et DCie [Badri 08]; TCC et LCC [Bieman 95]). Nous nous sommes vite aperçus, suite aux premiers résultats obtenus, que ces métriques ne donnaient pas de bons résultats en termes de prévision (valeurs de corrélation assez faibles), ce qui laissait croire qu'elles ne pouvaient pas être utilisées comme indicateur de présence de code pouvant correspondre à des préoccupations transverses. Suite à ces premières expérimentations, nous avons changé de démarche en nous intéressant plus au manque de cohésion (inverse de la cohésion pouvant renseigner la disparité dans le code). Pour faciliter alors l'évaluation et la comparaison entre ces différentes métriques, et sachant que les différentes versions considérées de LCOM sont plutôt des métriques de manque de cohésion (inverse par rapport aux métriques de cohésion) et quelles sont non normalisées, nous avons donc dérivé des métriques originales DCd, DCd, DCde, DCdi, TCC et LCC (pour le but de notre étude) des estimations du manque de cohésion (selon la démarche des LCOM) de chacune de ces métriques. Ce sont, en fait, ces versions des métriques indiquant plutôt le manque de cohésion qui ont été utilisées dans notre étude. L'un des buts principaux étant surtout d'évaluer si le manque de cohésion (disparité dans le code) pouvait être utilisé comme critère pouvant aider à déceler dans du code objet des parties de code contenant des préoccupations transverses. Ceci pourrait être utilisé

pour supporter l'aspect mining. L'idée était, aussi, d'évaluer de façon comparative les métriques de cohésion (manque de cohésion) pour voir laquelle ou lesquelles pouvaient être considérées comme étant de bons modèles prédictifs relativement aux autres.

V-2.3. Analyse du code : extraction des classes candidates

V-2.3.1. Fan-in analysis

L'outil d'analyse de code pour la recherche de classes candidates (contenant du code pouvant correspondre à des préoccupations transverses) utilisé dans notre recherche est le FINT, un plug-in Éclipse. Il supporte trois techniques d'aspect mining (par analyse de code) dont le Fan-in analysis. Cette dernière est la technique d'analyse fondamentale de l'outil. Elle détermine tous les appels entre les méthodes du système analysé. L'exécution de l'analyse Fan-in donne en résultat un arbre dont les nœuds sont les méthodes appelées et les branches les méthodes appelantes. Pour affiner le résultat, nous avons appliqué des filtres lors de l'analyse du code. L'outil inclut des filtres dans ses différentes techniques. Pour affiner les résultats, l'outil permet soit de fixer un seuil de fan-in soit d'appliquer les filtres de méthodes appelées ou appelantes. Dépendamment des objectifs de l'analyse, l'outil permet par exemple pour le cas du filtre des appelés de filtrer les méthodes accesseurs ainsi que les méthodes de librairie. Pour filtrer les appels et affiner nos résultats, nous avons fixé un seuil du fan-in. Sont donc exclues de notre étude les méthodes de fan-in inférieur à la moyenne (le seuil considéré dans notre expérimentation) : c'est le filtre des appelés. Les méthodes appelées ainsi filtrées sont nos aspects candidats. Les classes de ces méthodes seront considérées comme classes candidates à la restructuration. Le résultat est binaire : il prend la valeur 1 si la classe (considérée comme classe candidate) a un fan-in supérieur ou égal à la moyenne des fan-in et la valeur 0 sinon. Ainsi, de l'ensemble des projets analysés, nous avons obtenu les résultats suivants :

Freecs : 6 classes candidates sur 128 classes du projet, valeur seuil du fan-in égale à 18.

Jmol8 : 9 classes candidates sur 315 classes, valeur seuil du fan in égal à 11.

EJB : 18 classes candidates sur 1721 classes, valeur seuil du fan-in égal à 18.

Jflex : 8 classes candidates sur 45 classes, valeur seuil du fan in égal à 9.

Lucene : 83 classes candidates sur 322 classes, valeur seuil du fan in égal à 11.

JasperReports : 107 classes candidates sur 1177 classes, valeur seuil du fan in égal à 22.

PMD : 27 classes candidates sur 642 classes, valeur seuil du fan in égal à 10.

JhotDraw : 40 classes candidates sur 343 classes, valeur seuil du fan in égal à 14.

V-2.3.2. Grouped calls analysis

Le Grouped calls ou l'analyse des appels groupés nécessite, avant son exécution, que l'analyse Fan-in soit exécutée (que le modèle construit par le Fan-in existe déjà). Pour exécuter le grouped calls analysis, il suffit d'aller sur la vue concernée, et cliquer sur le bouton « rafraîchir le modèle ». Le résultat s'affiche après analyse du code exactement comme dans l'analyse précédente, à la seule différence que les nœuds (aspects candidats) sont constitués des groupes de méthodes qui partagent les mêmes appelants et les branches sont ces appelants. Ce résultat peut également être trié et filtré exactement comme dans la technique Fan-in. Le grouped calls analysis fournit le filtre des méthodes set et get, des méthodes de librairies, et aussi des utilitaires. Il faut fixer également le seuil des appelants (dans la plupart des cas, c'est le seuil du Fan-in), et le nombre minimum de méthodes par groupe d'appelés.

Nous avons expérimenté cette technique juste pour comparer nos résultats avec ceux existants dans la littérature dans le cas unique de Jhotdraw; ce qui a donné une bonne

similitude. Nous avons aussi examiné la combinaison des deux techniques toujours dans le cadre de nos vérifications. Nous avons fait le constat suivant : en utilisant le même seuil pour les deux techniques (ce qui est recommandé), leur combinaison est confondue au Grouped calls analysis qui, après analyse des résultats, est une partie des résultats du Fan-in. Alors, dans le cadre de notre étude, et dans le souci de capturer le maximum d'informations afin de vérifier notre hypothèse, nous avons utilisé la technique Fan-in.

V-3. Statistiques descriptives des systèmes utilisés

Le tableau V donne quelques métriques de base caractérisant les systèmes analysés. Des huit systèmes analysés, nous avons au total environ 350 000 lignes de code (métrique LOC dans le tableau), 4 000 classes (#classes), et 30 000 méthodes (#méthodes). Pour affiner les résultats de l'analyse par l'outil d'aspect mining (FINT), nous avons exclu de notre analyse les méthodes abstraites, les méthodes de librairies, d'interfaces, et les constructeurs. La classe « candidate » d'après la technique utilisée est la classe ayant un fan-in supérieur au seuil. Le seuil retenu pour l'analyse de chaque système sélectionné est la moyenne des fan-in des différentes méthodes du système. Le tableau VI résume les statistiques descriptives des différents projets analysés pour les métriques sélectionnées.

Tableau V Les métriques de base pour les systèmes sélectionnés.

Systèmes	LOC	#Classes	#Méthodes
EJB	94776	1721	10523
FreeCS1.2.2	15141	128	848
JasperReports	110717	1177	10418
Jflex1.3.1	8447	45	347
Jmol8	30288	315	1731
Lucene2.2.0	22940	322	1616
PMD	46155	642	3888
JhotDraw	19144	343	1772

Tableau VI: Statistiques descriptives des métriques sélectionnées (Min/Max/Moyenne).

Metr.\Syst	EJB	FreeCS	JasperReports	JFlex	Jmol8	Lucene	PMD	JhotDraw
LDCd	0/1/0.162	0/1/0.097	0/1/0.113	0/1/0.183	0/1/0.187	0/1/0.187	0/1/0.075	0/1/0.184
LDCi	0/1/0.166	0/1/0.101	0/1/0.117	0/1/0.193	0/1/0.195	0/1/0.198	0/1/0.75	0/1/0.193
LDCde	0/1/0.198	0/1/0.131	0/1/0.143	0/1/0.237	0/1/0.244	0/1/0.218	0/1/0.088	0/1/0.222
LDCie	0/1/0.208	0/1/0.142	0/1/0.152	0/1/0.272	0/1/0.272	0/1/0.239	0/1/0.090	0/1/0.243
L-TCC	0/1/0.143	0/1/0.094	0/1/0.107	0/1/0.175	0/1/0.178	0/1/0.168	0/1/0.067	0/1/0.149
L-LCC	0/1/0.147	0/1/0.098	0/10.110	0/1/0.183	0/1/0.185	0/1/0.177	0/1/0.067	0/1/0.156
LCOM1	0/5640/ 47.663	0/3514/ 75.650	0/13970/ 306.074	0/674/ 52.932	0/32168/ 336.961	0/2445/ 60.184	0/119700/ 571.634	0/940/ 34.060
LCOM2	0/100/ 41.892	0/100/ 77.562	0/100/71.122	0/99/53.979	0/100/ 62.556	0/100/ 53.950	0/100/ 58.189	0/100/ 41.268
LCOM3	0/100/ 39.230	0/100/ 72.542	0/100/70.134	0/97/ 53.821	0/100/ 72.139	0/100/ 50.113	0/100/ 52.099	0/100/ 19.364

V-4. Approche statistique

Avant d'utiliser les méthodes d'analyse avancées, comme la régression logistique par exemple, il est nécessaire dans un premier temps, de découvrir les données afin d'identifier des tendances, de repérer des anomalies ou tout simplement de disposer d'informations essentielles telles que le minimum, le maximum, ou la moyenne d'un échantillon de données.

XLSTAT nous propose un nombre important de statistiques descriptives et de graphiques tels que les diagrammes en boîte qui nous permettent d'avoir un premier aperçu pertinent de nos données.

V-4.1. Diagramme en boîte ou boxplot

Les diagrammes en boîtes et moustaches sont des graphiques uni-variés d'échantillons de données quantitatives. Ils nous permettent de visualiser les tendances générales de la variable «classe candidate » par rapport à sa valeur de la cohésion. C'est une représentation simple et assez complète puisque dans la version proposée par XLSTAT sont affichés le minimum, le 1er quartile (Q1), la médiane, la moyenne, le 3ième quartile (Q3), ainsi que les deux limites (les extrémités des « moustaches ») au-delà desquelles on peut considérer que les valeurs sont anormales. La moyenne est affichée sous la forme du signe "+", et la médiane sous la forme d'une ligne noire. Les limites sont ainsi calculées :

Limite inférieure : liminf = X(i) tel que $\{X(i) - [Q1 - 1.5\ (Q3 - Q1)]\}$ soit minimal

et $X(i) = Q1 - 1.5\ (Q3 - Q1)$. (6)

Limite supérieure : limsup = X(i) tel que {X(i) - [Q3 + 1.5 (Q3 – Q1)]} soit minimal

et X(i) = Q3 + 1.5 (Q3 – Q1). (7)

Les valeurs en dehors de l'intervalle] Q1 - 3 (Q3 – Q1); Q3 + 3 (Q3 – Q1) [sont affichées avec le symbole "*".

Celles comprises dans [Q1 - 3 (Q3 – Q1); Q1 – 1.5 (Q3 – Q1)] ou

[Q3 + 1.5 (Q3 – Q1); Q3 + 3 (Q3 – Q1)] sont affichées avec le symbole "o".

V-4.2. Régression logistique

L'objectif de ce travail étant d'évaluer jusqu'à quel degré le manque de cohésion peut être utilisé comme un indicateur de classe candidate. Pour cela, il nous faut démontrer que les classes détectées d'après l'outil d'aspect mining comme candidates, outil validé dans le domaine de l'aspect mining, ont une faible cohésion, donc présentent une disparité dans leur code. Nous recherchons donc une corrélation entre les résultats de l'outil et les valeurs des métriques de cohésion des différentes classes du projet. Pour y arriver, nous avons utilisé XLSTAT [XLSTAT 07] pour analyser les données et avons focalisé notre attention sur l'une de ses méthodes de modélisation des données : la régression logistique. La régression logistique est une méthode qui permet de modéliser les variables binaires ou des sommes de variables binaires. Elle est très utilisée dans divers domaines tels que le domaine médical (guérison ou non d'un patient), en sociologie et en épidémiologie (analyse d'enquêtes), en marketing quantitatif (achat ou non de produits ou services suite à une action) et en finance pour la modélisation de risques (scoring).

Elle est généralement utilisée lorsqu'on veut décider relativement à la survenance ou non d'un évènement donné. Dans le cas de notre étude, nous voulons décider sur la survenance de l'évènement « une classe faiblement cohésive contient du code correspondant à des préoccupations transverses ». Autrement dit, à partir de cette technique de modélisation de données, nous voulons évaluer jusqu'à quel degré, la cohésion peut-elle être utilisée pour identifier les préoccupations transverses dans un code Java, et supporter ainsi l'aspect mining.

La régression logistique appartenant à la famille des modèles GLM (Generalized Linear Models), s'adapte très bien à la problématique traitée dans ce projet; dans ce sens qu'elle relie un événement (classe candidate) à une combinaison linéaire de variables explicatives discontinues (les métriques de cohésion). De plus, cette méthode est bien soutenue par une analyse exploratoire préliminaire donnée par les box-plots.

Pour la régression logistique, la variable dépendante, aussi appelée variable réponse (classe candidate), suit une loi de Bernoulli de paramètre p (p la probabilité moyenne pour que l'événement se produise), lorsque l'expérience est répétée une fois, ou une loi Binomiale (n, p) si l'expérience est répétée n fois (par exemple la même dose est essayée sur n insectes). Le paramètre de probabilité p est ici une fonction d'une combinaison linéaire des variables explicatives. Les fonctions les plus couramment utilisées pour relier la probabilité p aux variables explicatives sont la fonction logistique (on parle alors de modèle Logit) et la fonction de répartition de la loi normale standard (on parle alors de modèle Probit). Ces deux fonctions sont parfaitement symétriques et sigmoïdes.

La connaissance de la loi de distribution de l'événement étudié, permet d'écrire la vraisemblance de l'échantillon. Pour estimer les paramètres β du modèle (les coefficients de la fonction linéaire), on cherche à maximiser la fonction de vraisemblance. Contrairement aux autres méthodes comme la régression linéaire, une solution analytique exacte n'existe pas. Il est donc nécessaire d'utiliser un algorithme

itératif. XLSTAT utilise un algorithme de Newton-Raphson. L'utilisateur peut modifier s'il le souhaite le nombre maximum d'itérations et le seuil de convergence.

XLSTAT donne la possibilité d'afficher le tableau de classification (aussi appelé matrice de confusion) qui permet de calculer un pourcentage d'observations bien classées, pour un point de séparation (cutoff) donné. Typiquement, pour une valeur de 0.5 du point de séparation, si la probabilité est inférieure à 0.5, l'observation est considérée comme étant affectée à la classe 0, sinon, elle est affectée à la classe 1.

La courbe ROC peut aussi être affichée. La courbe ROC (Receiver Operating Characteristics) permet de visualiser la performance d'un modèle, et de la comparer à celle d'autres modèles. Les termes utilisés viennent de la théorie de détection du signal. On désigne par sensibilité (sensivity) la proportion d'événements positifs bien classés. La spécificité (specificity) correspond à la proportion d'événements négatifs bien classés. Si l'on fait varier la probabilité, seuil à partir de duquel on considère qu'un événement doit être considéré comme positif, la sensibilité et la spécificité varient. La courbe des points (1-spécificité, sensibilité) est la courbe ROC. L'aire sous la courbe (ou Area Under the Curve – AUC) est un indice synthétique calculé pour les courbes ROC.

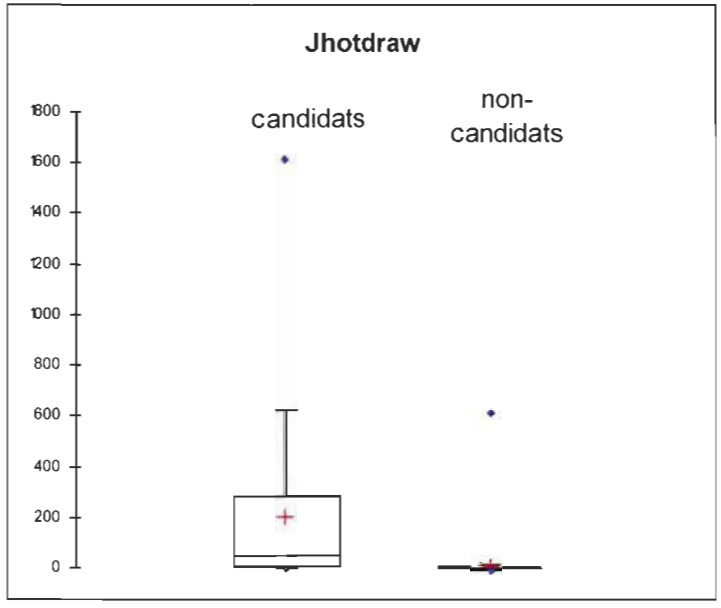
L'AUC correspond à la probabilité pour qu'un événement positif ait une probabilité donnée par le modèle plus élevée qu'un événement négatif. Pour un modèle idéal, on à AUC=1, pour un modèle aléatoire, on a AUC=0.5. On considère habituellement que le modèle est bon dès lors que la valeur de l'AUC est supérieure à 0.7. Un modèle bien discriminant doit avoir une AUC entre 0.87 et 0.9. Un modèle ayant une AUC supérieure à 0.9 est dit excellent.

V-5. Résultats et interprétations

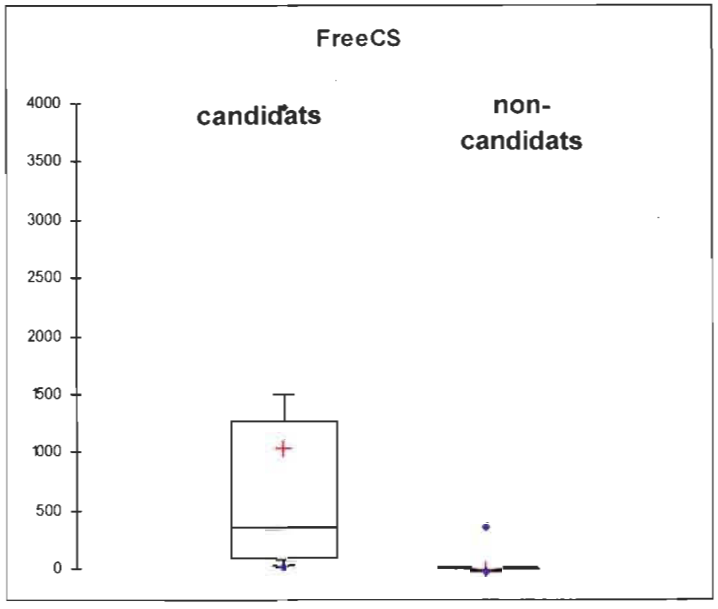
V-5.1. Diagramme en boite ou boxplot

Les diagrammes donnés dans les figures qui suivent nous donnent les tendances globales de notre expérimentation. En observant les boites, on peut constater, que dans l'ensemble, les valeurs des variables (candidate et non-candidate) sont comprises dans l'espace des valeurs normales (entre limInf et limSup). Nous pouvons observer également que les boîtes sont moins aplaties pour les classes qui présentent des aspects candidats que pour celles qui n'en présentent pas. En conséquence, les moyennes des candidats pour chacune des métriques sélectionnées et pour l'ensemble des projets expérimentés sont plus élevées pour les classes candidates que celles non-candidates (avoisinant zéro). Ceci peut s'interpréter comme suit : les classes candidates dans l'ensemble présentent un manque de cohésion important (disparité dans le code) que celle non-candidates. Nous remarquons, également, que les systèmes comme JhotDraw, FreeCS, Jflex et PMD, avec en moyenne plus de 35% des classes détectées comme candidates, ont la majorité des valeurs candidates dans l'espace des valeurs normales. En résumé, ces tendances sont beaucoup plus accentuées pour des systèmes avec un grand nombre de classes candidates.

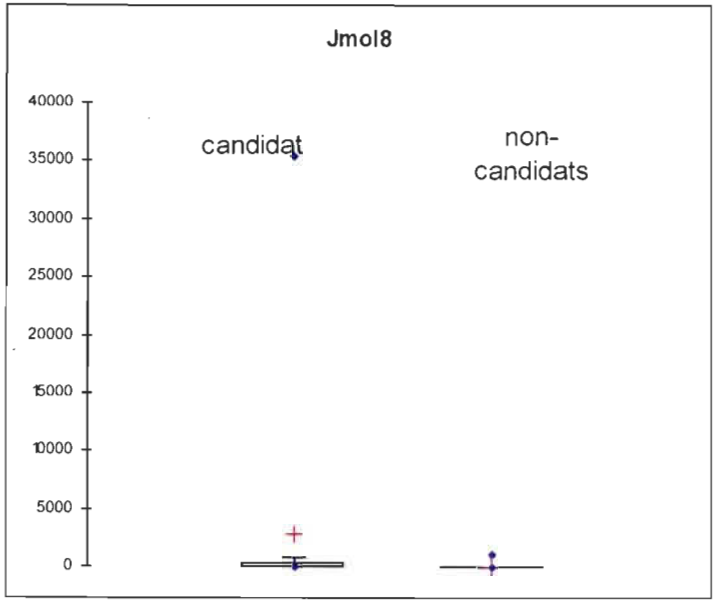
- **La métrique LDCd**



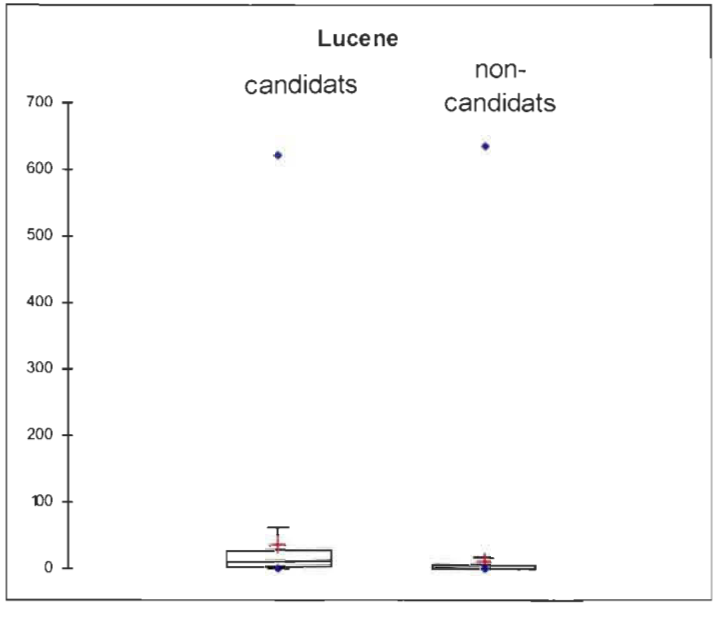
Boxplot JhotDraw.



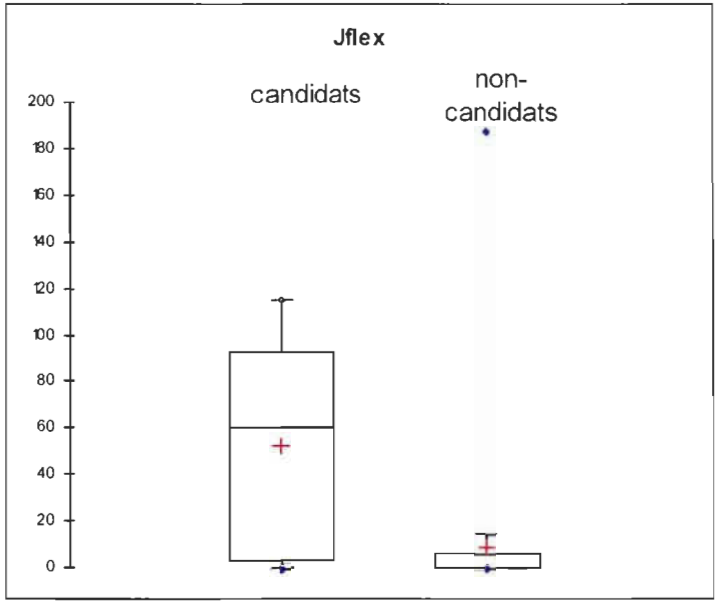
Boxplot FreeCS.



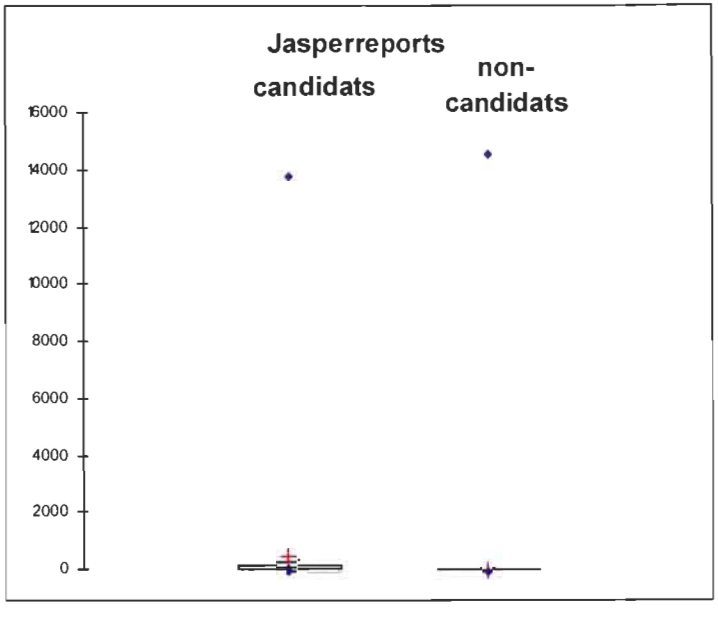
Boxplot Jmol8.



Boxplot Lucene.



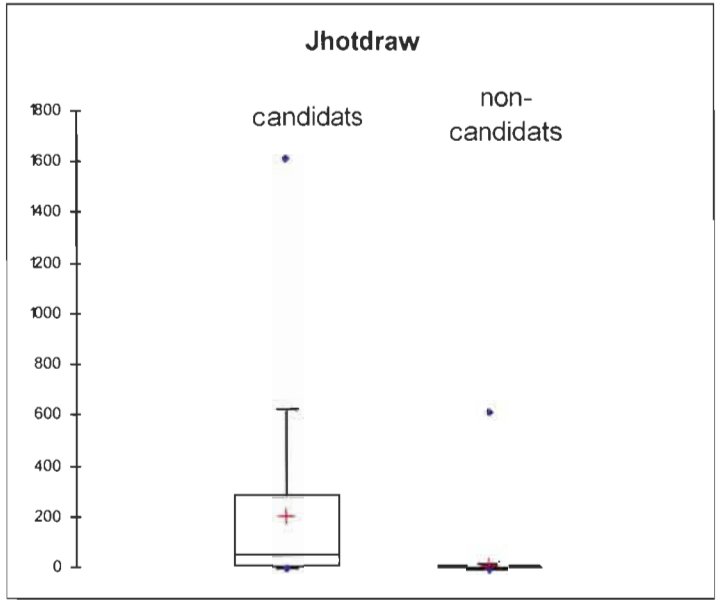
Boxplot Jflex.



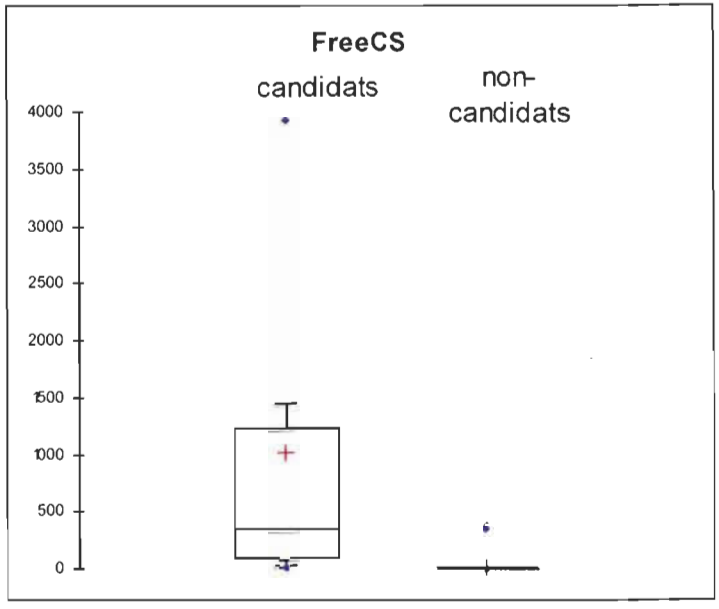
Boxplot JasperReports.

Figure 10: Boxplot LDCd / Jhot, FreeCS, Jmol8, Lucene, Jflex, Jasper.

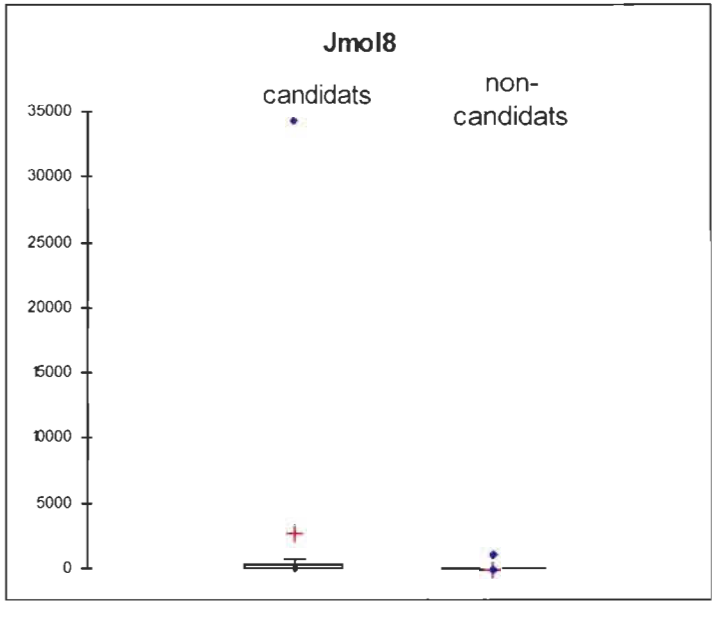
- **La métrique LDCi**



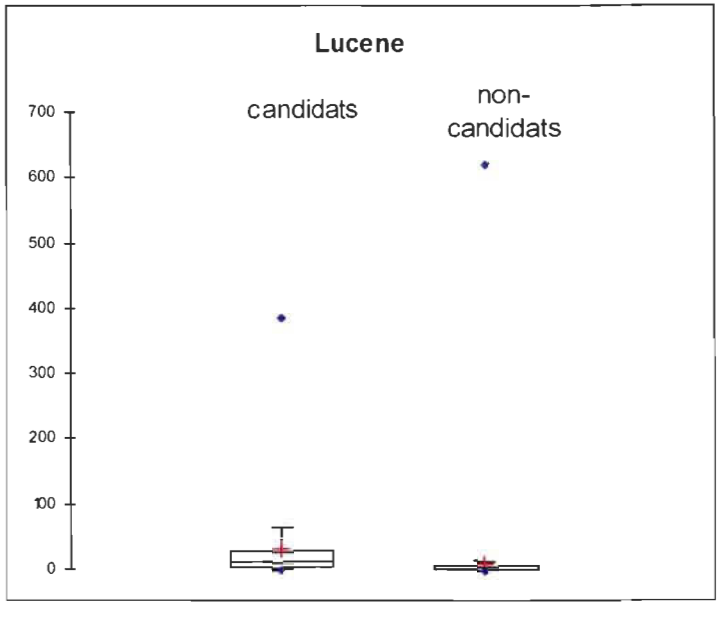
Boxplot JhotDraw.



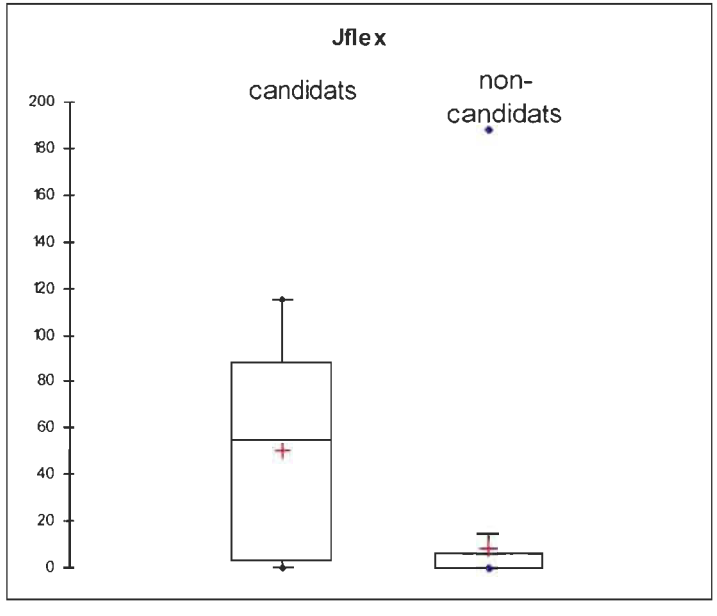
Boxplot FreeCS.



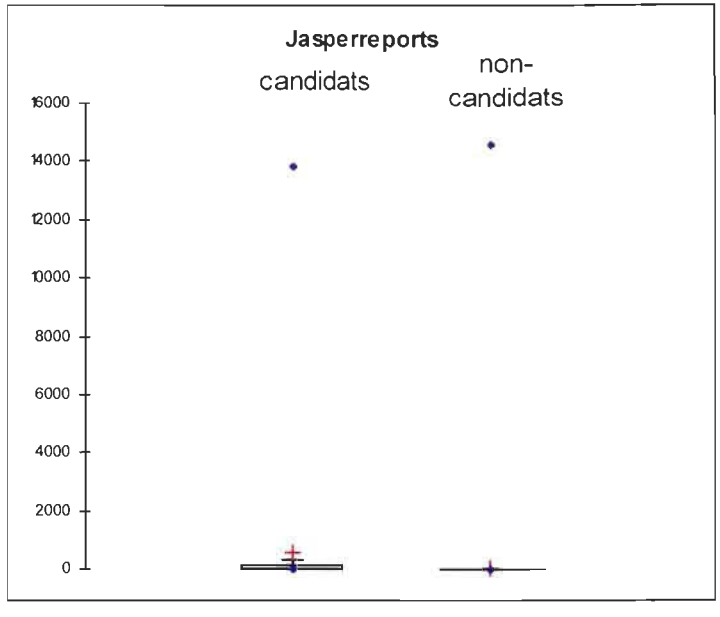
Boxplot Jmol8.



Boxplot Lucene.



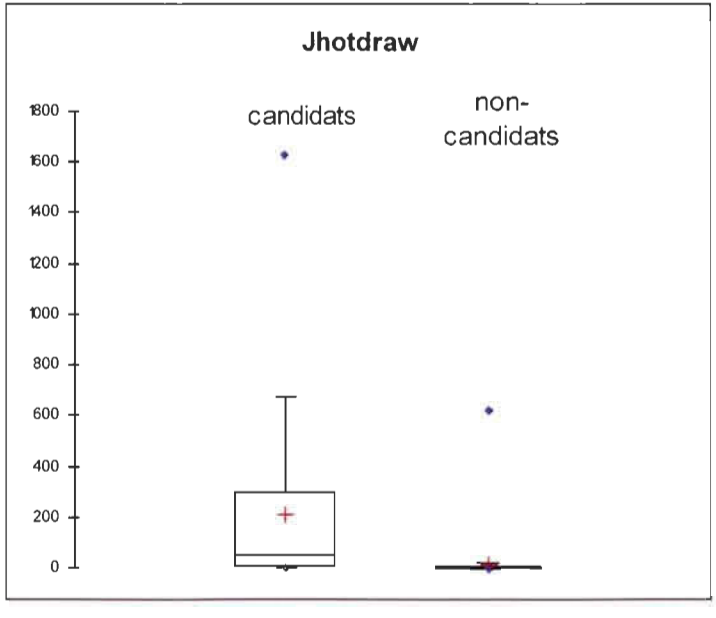
Boxplot Jflex.



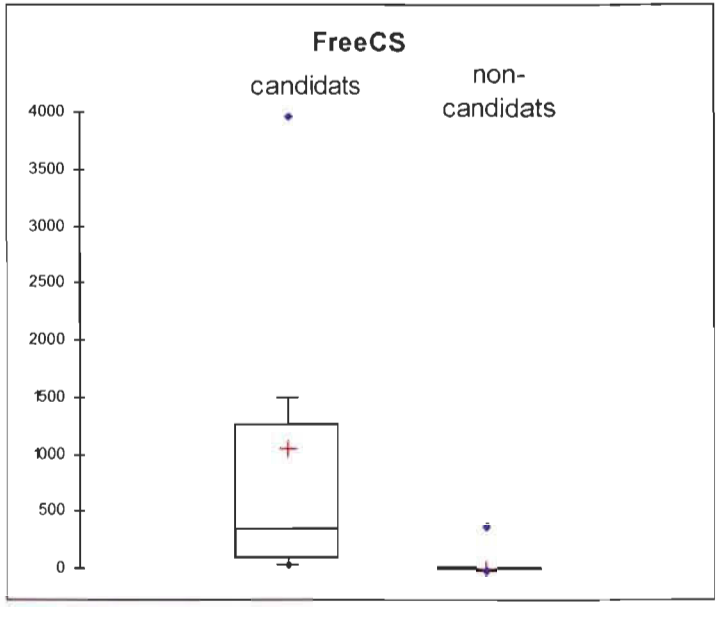
Boxplot JasperReports.

Figure 11: Boxplot LDCi / Jhot, FreeCS, Jmol8, Lucene, Jflex, Jasper.

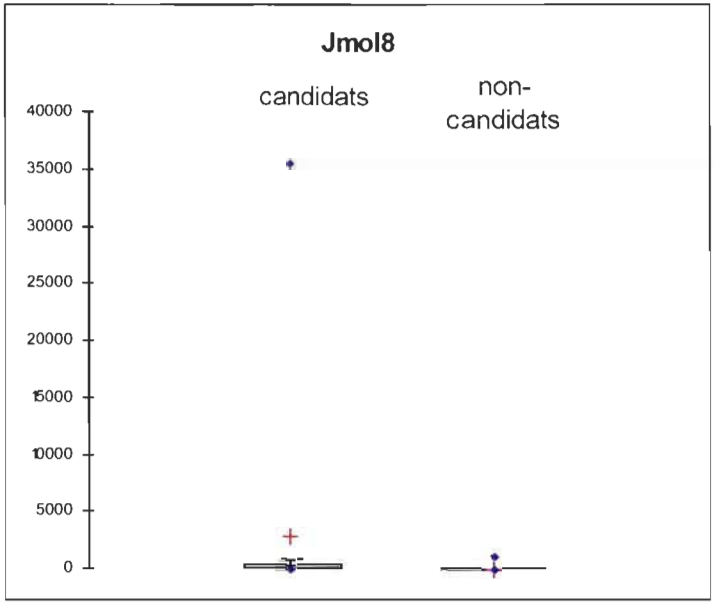
- **La métrique L-TCC**



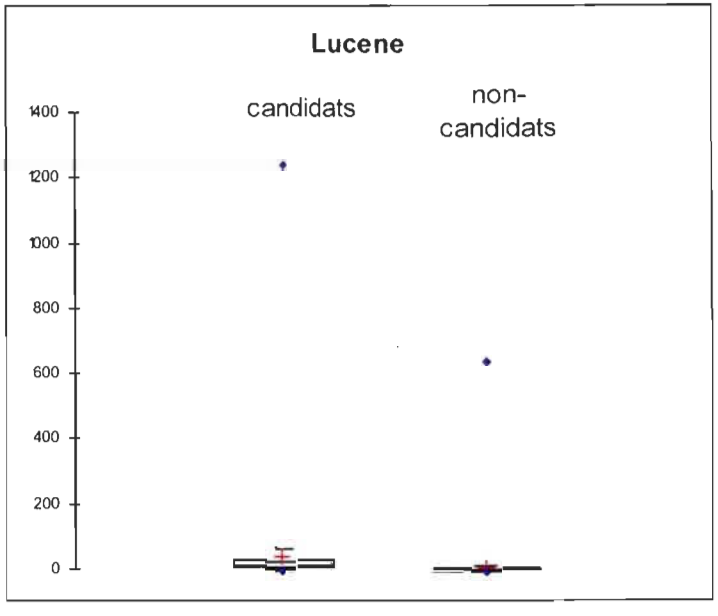
Boxplot JhotDraw.



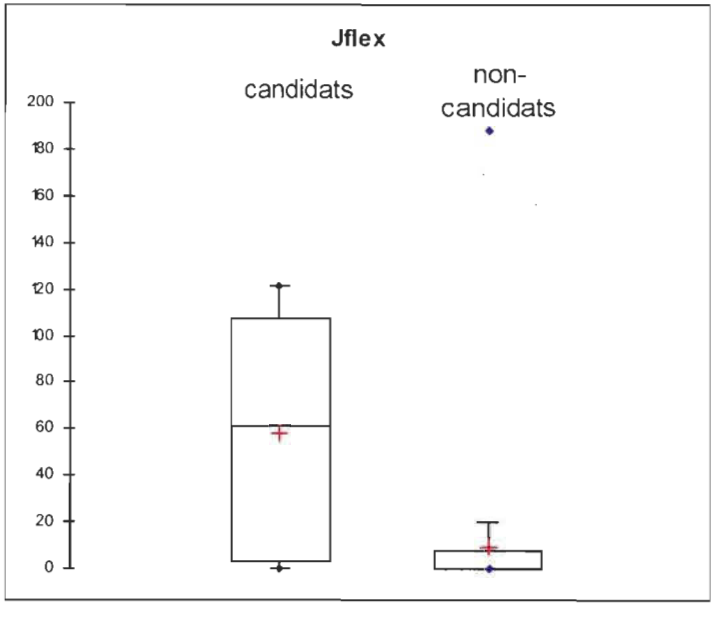
Boxplot FreeCS.



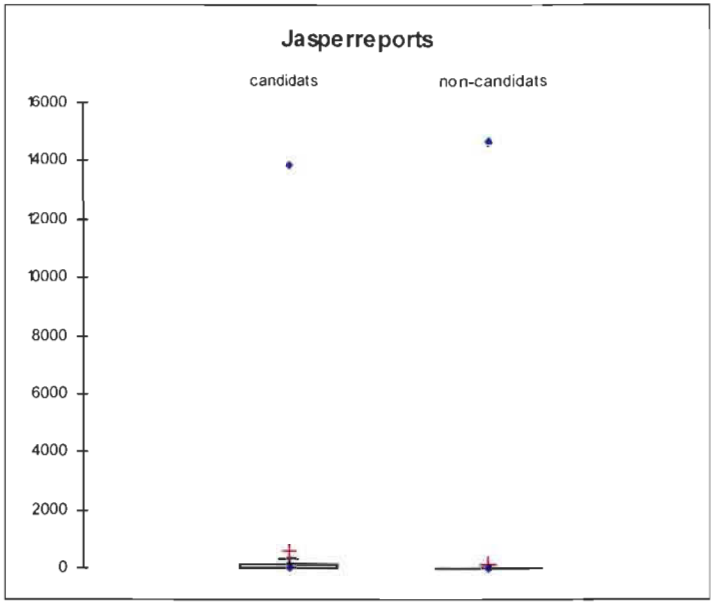
Boxplot Jmol8.



Boxplot Lucene.



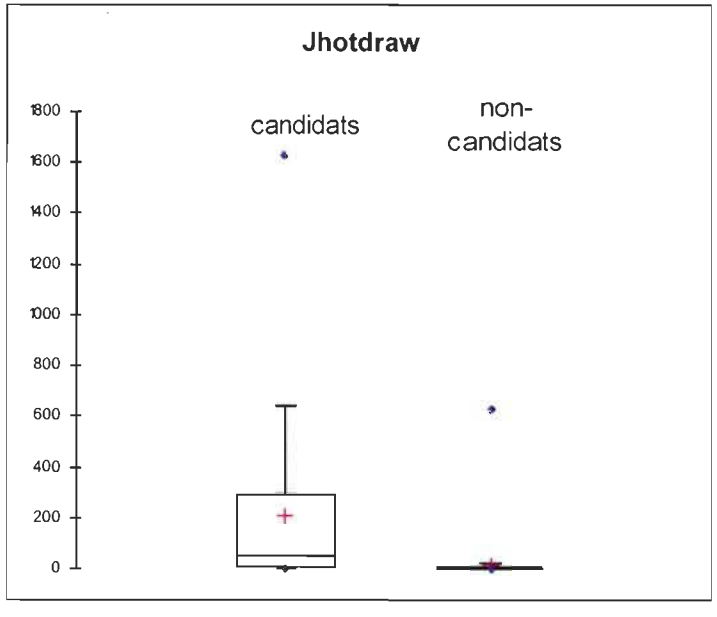
Boxplot Jflex.



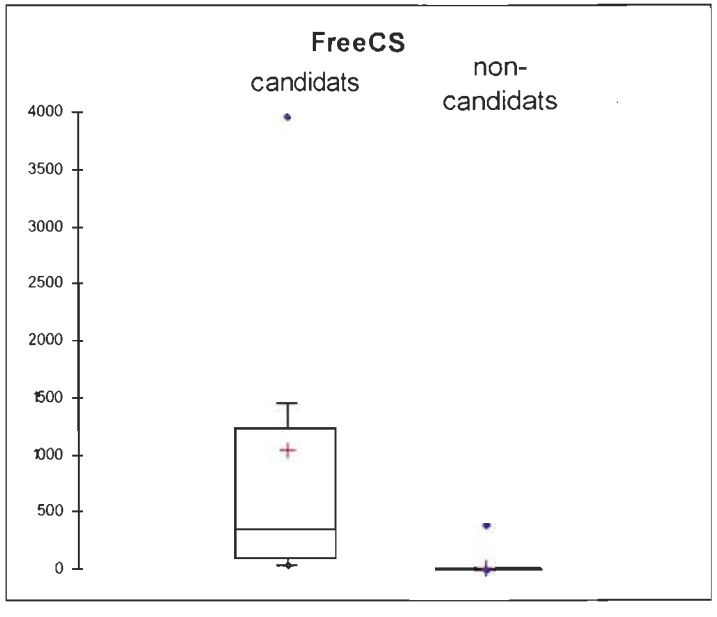
Boxplot JasperReports.

Figure 12: Boxplots L-TCC / Jhot, FreeCS, Jmol8, Lucene, Jflex, Jasper.

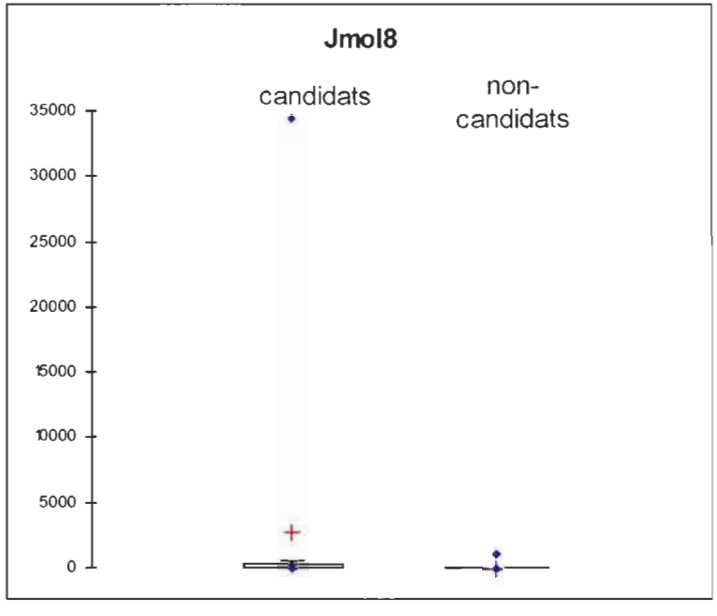
- **La métrique L-LCC**



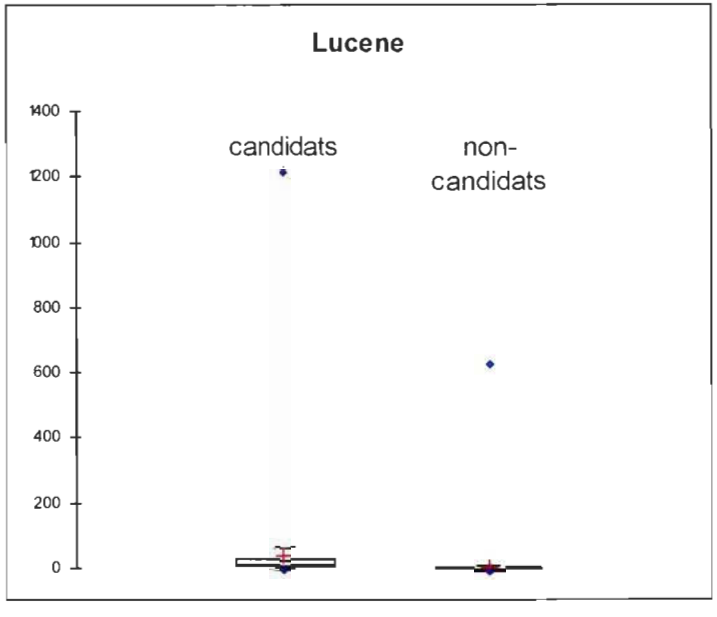
Boxplot JhotDraw.



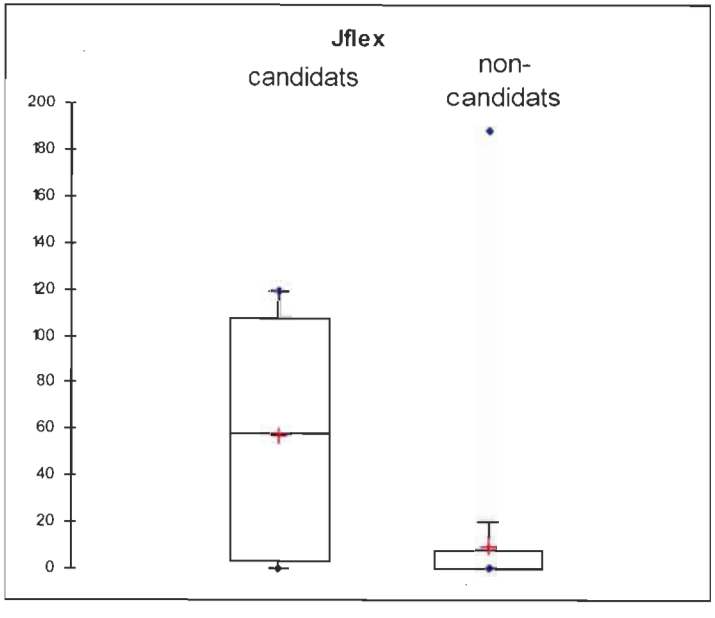
Boxplot FreeCS.



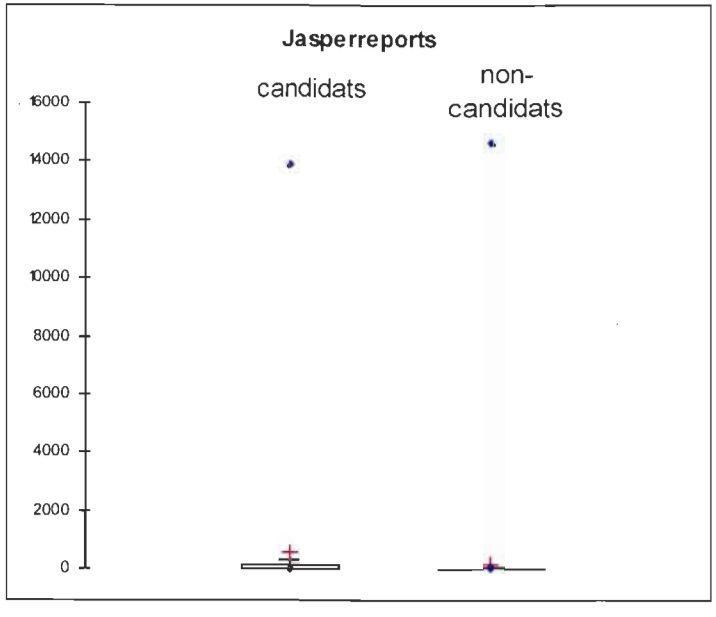
Boxplot Jmol8.



Boxplot Lucene.



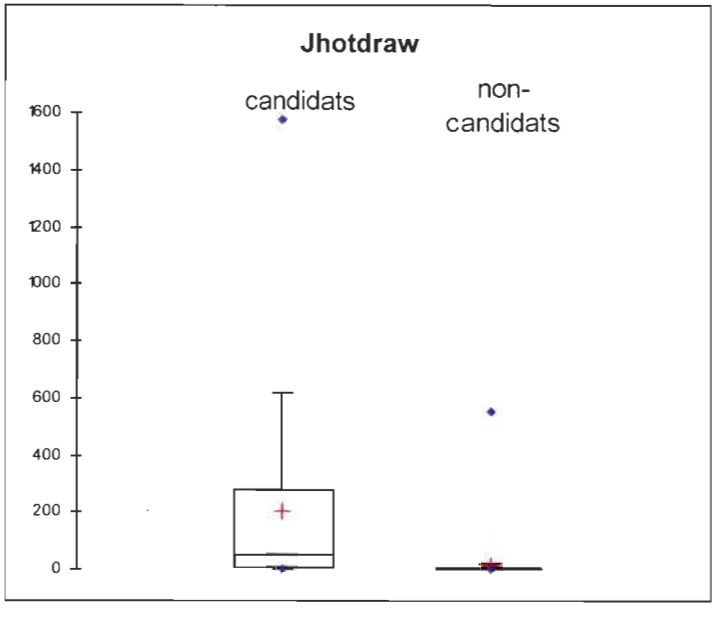
Boxplot Jflex.



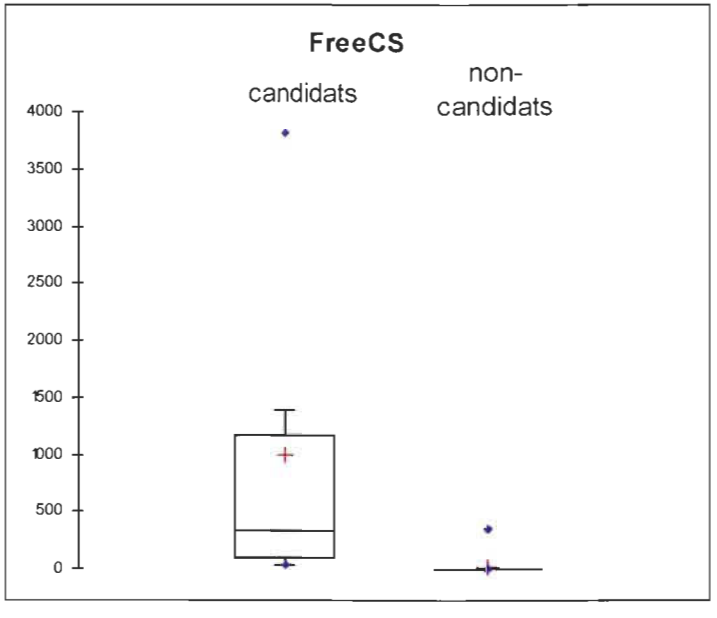
Boxplot JasperReports.

Figure 13: Boxplots L-LCC / Jhot, FreeCS, Jmol8, Lucene, Jflex, Jasper.

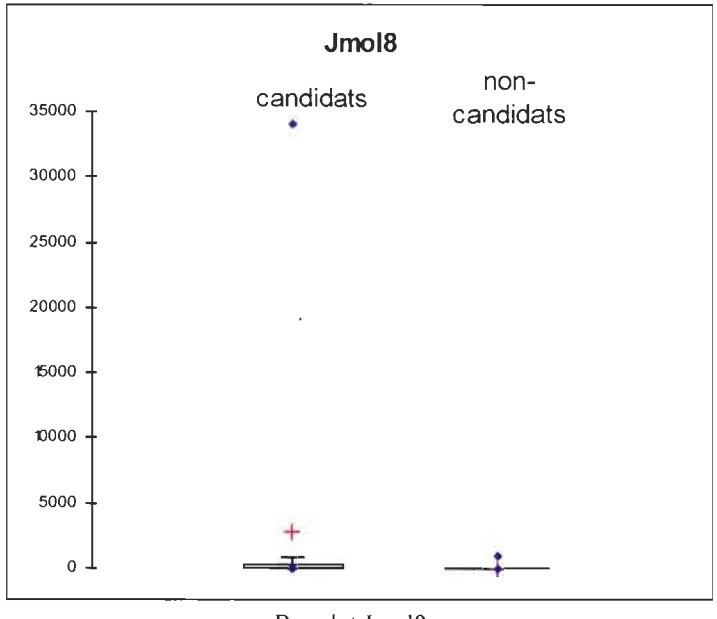
- **La métrique LDCde**



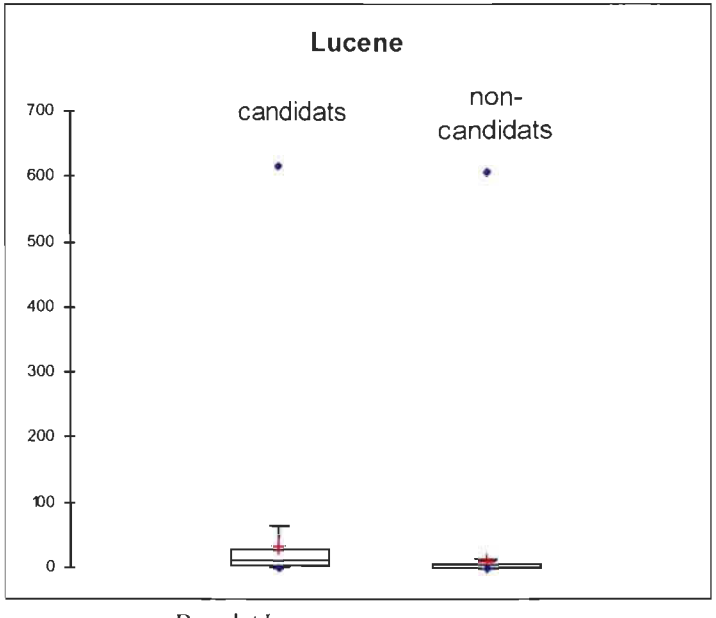
Boxplot JhotDraw.



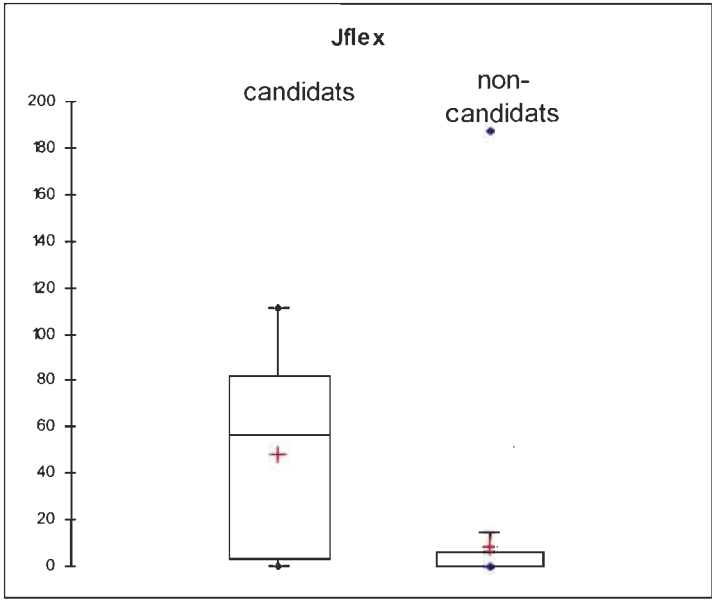
Boxplot FreeCS.



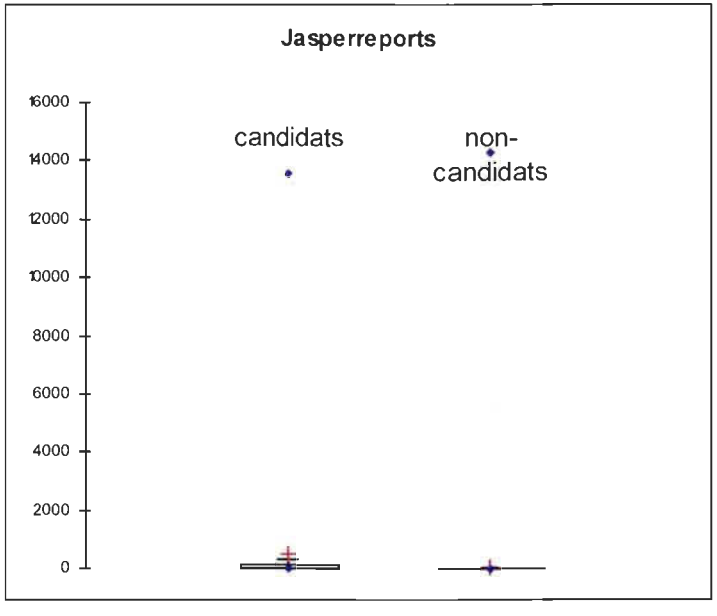
Boxplot Jmol8.



Boxplot Lucene.



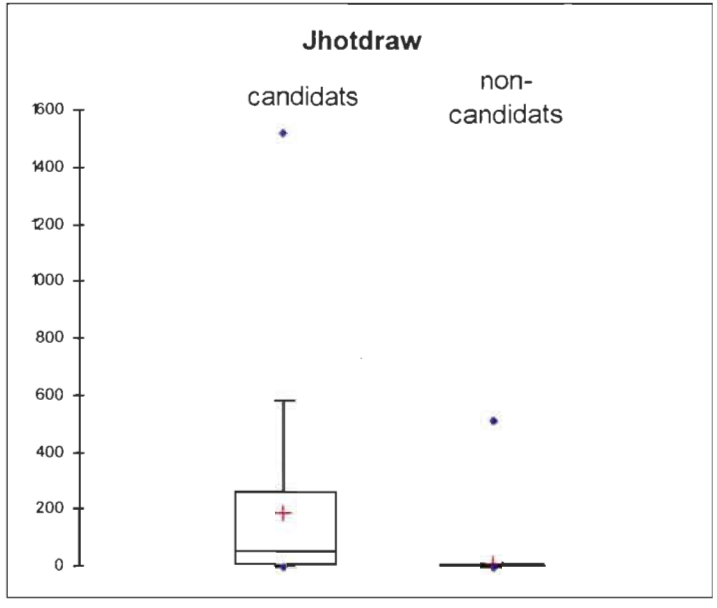
Boxplot Jflex.



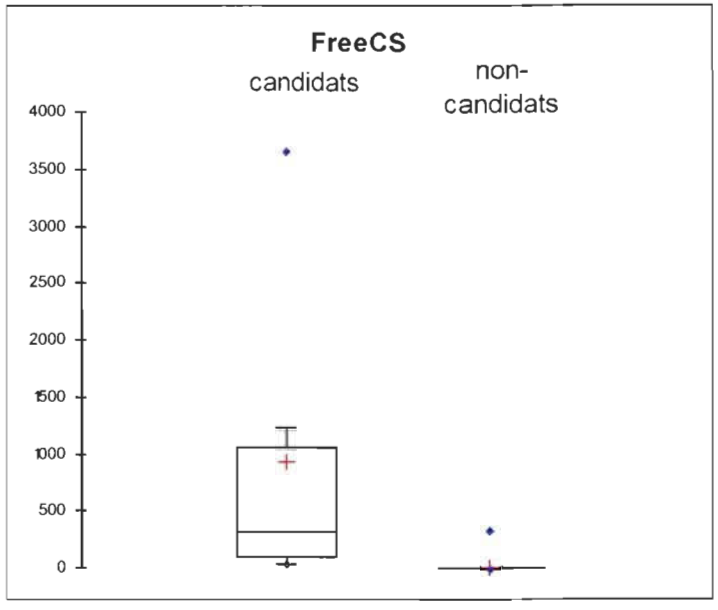
Boxplot JasperReports.

Figure 14: Boxplots LDCde / Jhot., FreeCS, Jmol8, Lucene, Jflex, Jasper.

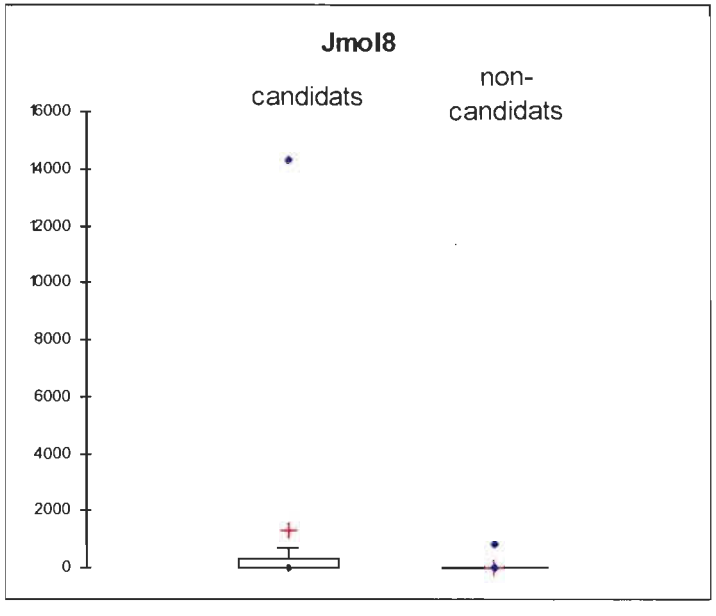
- **La métrique LDCie**



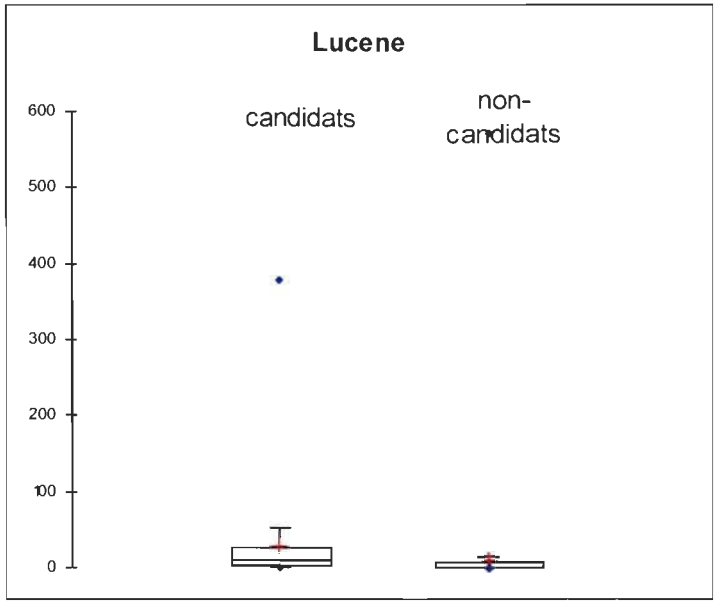
Boxplot JhotDraw.



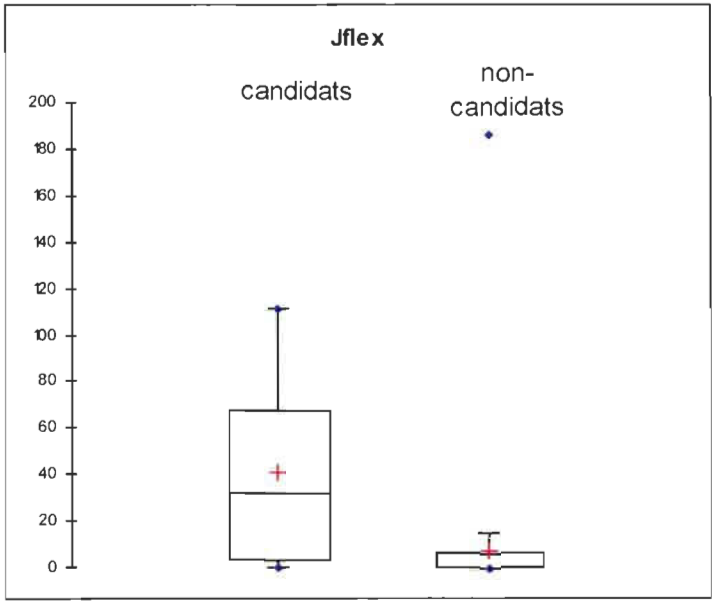
Boxplot FreeCS.



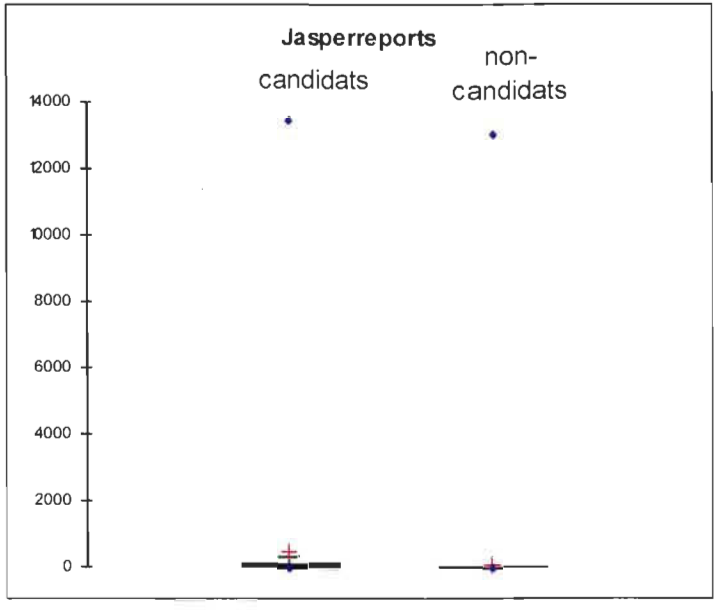
Boxplot Jmol8.



Boxplot Lucene.



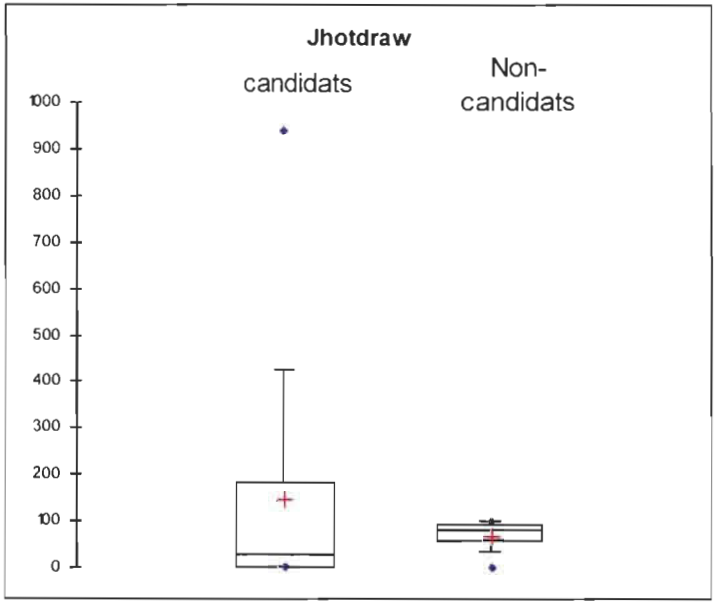
Boxplot Jflex.



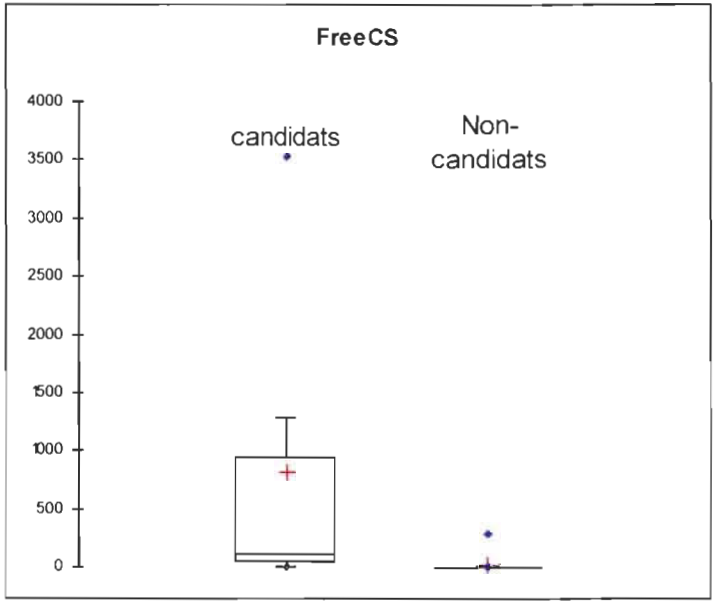
Boxplot JasperReports.

Figure 15: Boxplots LDCie /Jhot., FreeCS, Jmol8, Lucene, Jflex, Jasper.

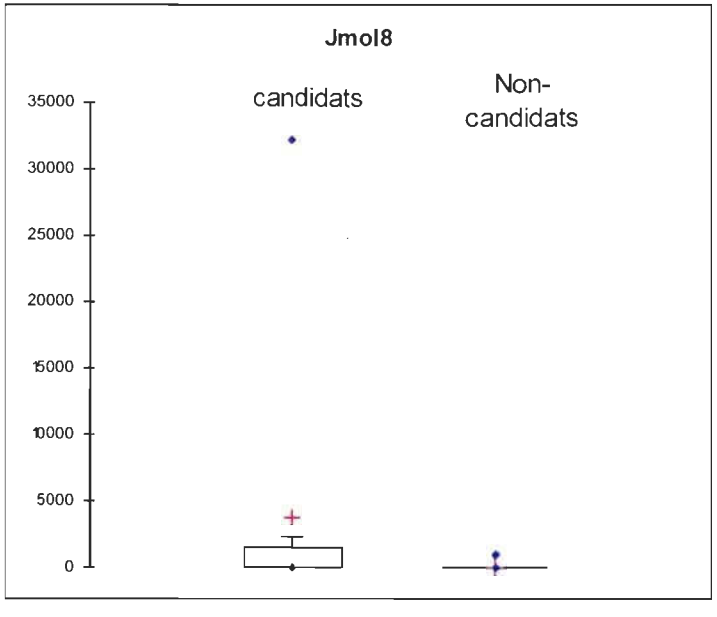
- **La métrique LCOM1**



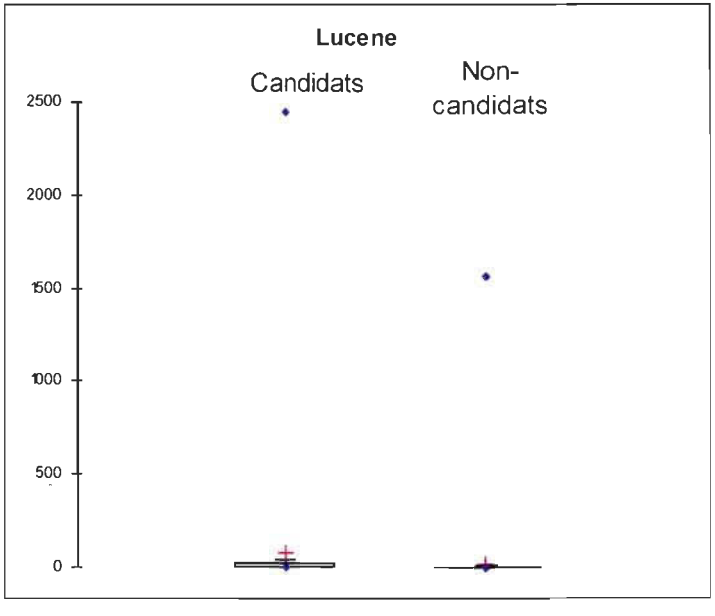
Boxplot JhotDraw.



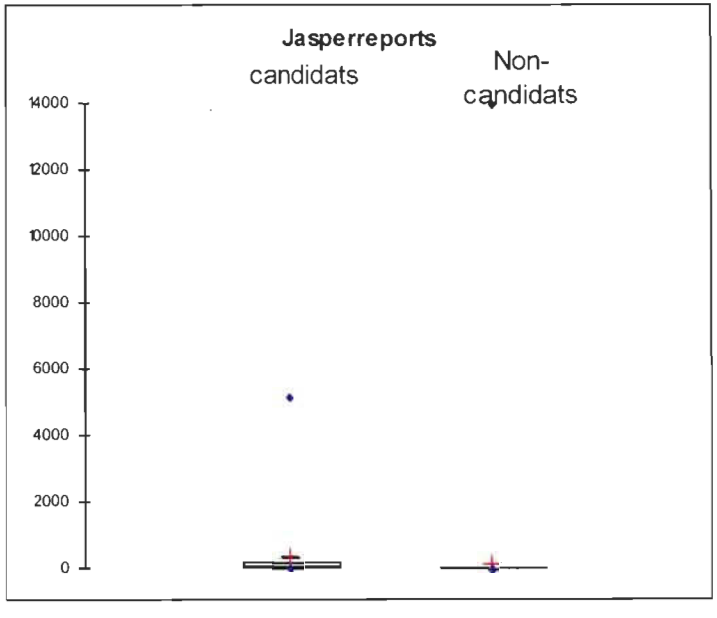
Boxplot FreeCS.



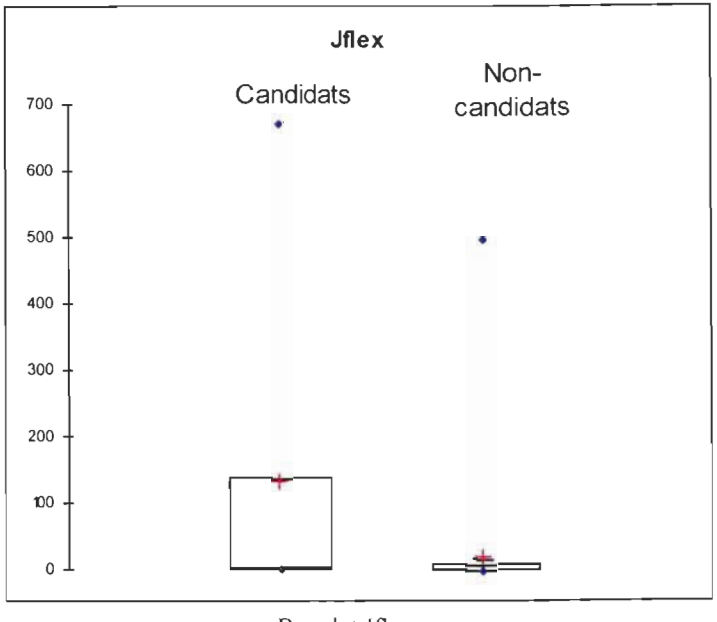
Boxplot Jmol8.



Boxplot Lucene.



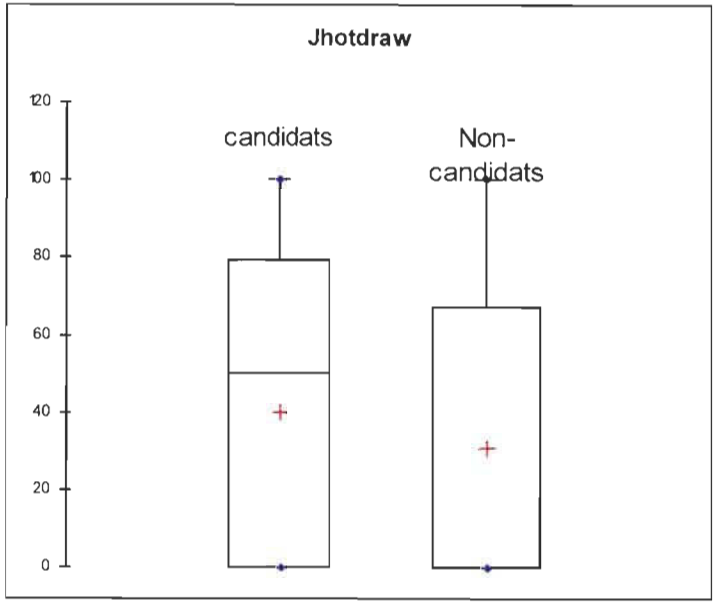
Boxplot JasperReports.



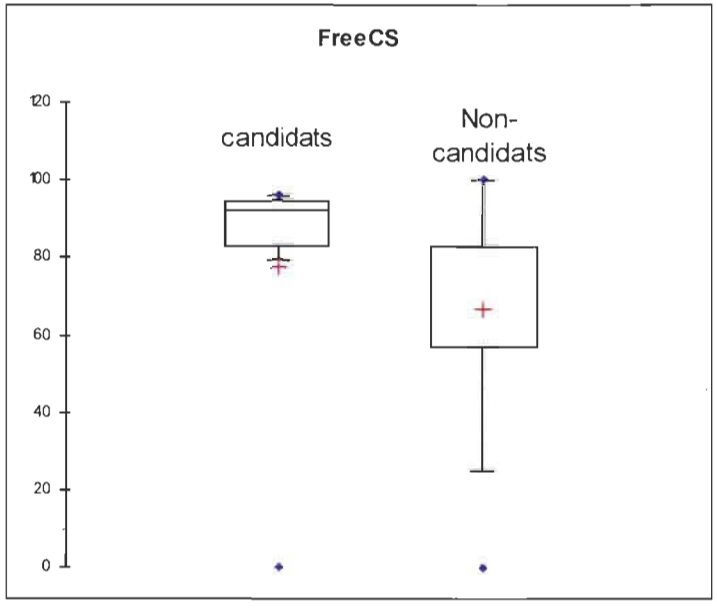
Boxplot Jflex.

Figure 16: Boxplots LCOM1 / Jhot., FreeCS, Jmol8, Lucene, Jflex, Jasper.

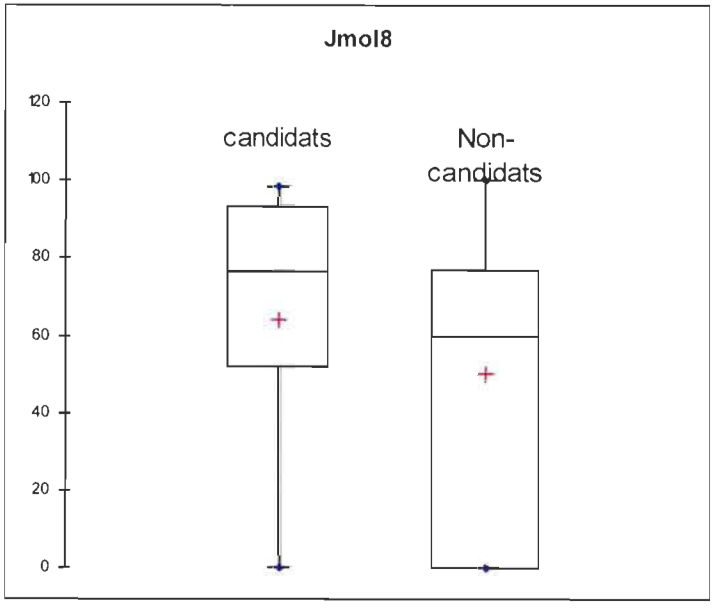
- **La métrique LCOM2**



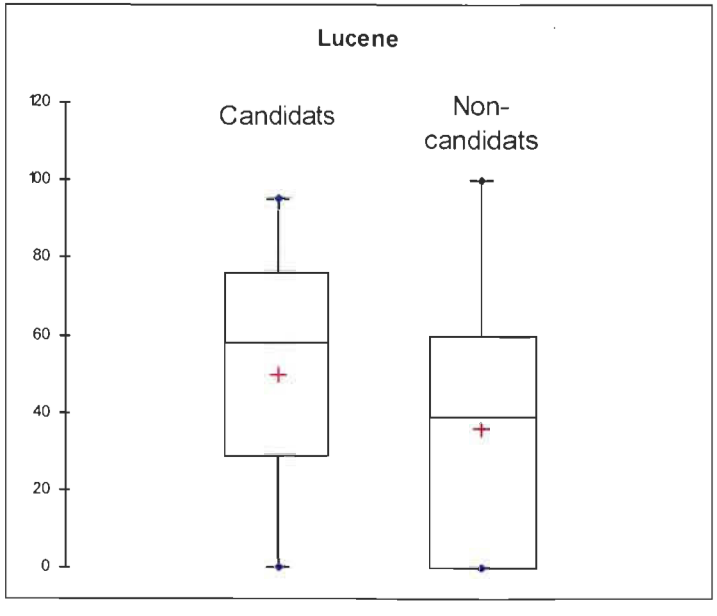
Boxplot JhotDraw.



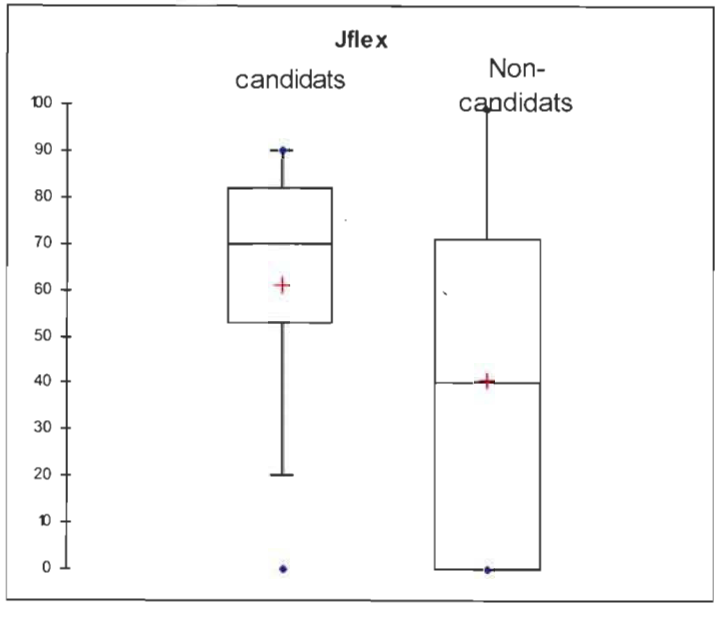
Boxplot FreeCS.



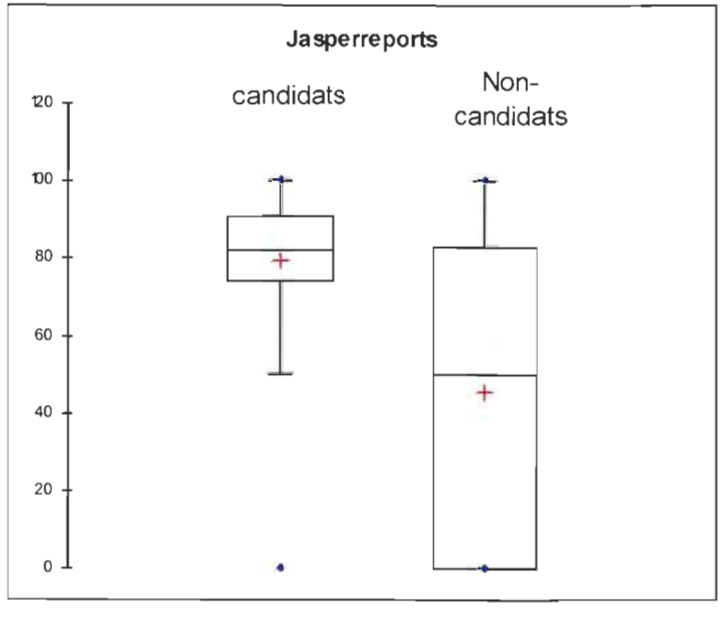
Boxplot Jmol8.



Boxplot Lucene.



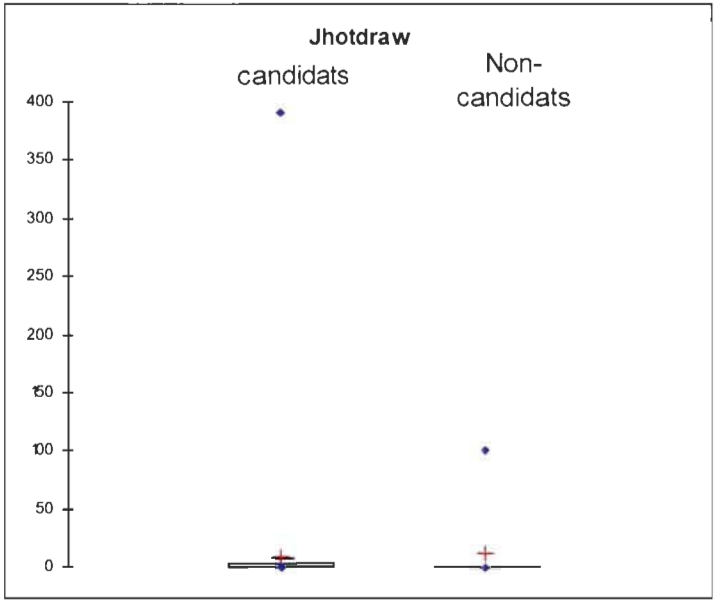
Boxplot Jflex.



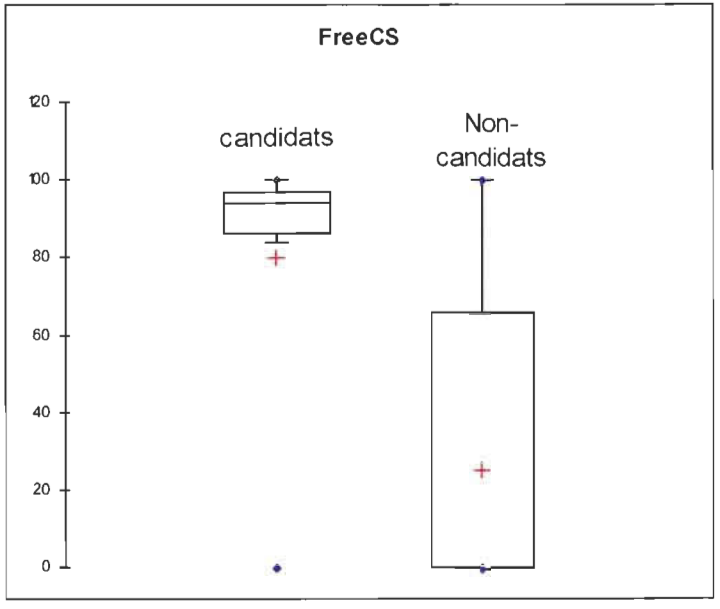
Boxplot JasperReports.

Figure 17: Boxplots LCOM2 / Jhot., FreeCS, Jmol8,Lucene, Jflex, Jasper.

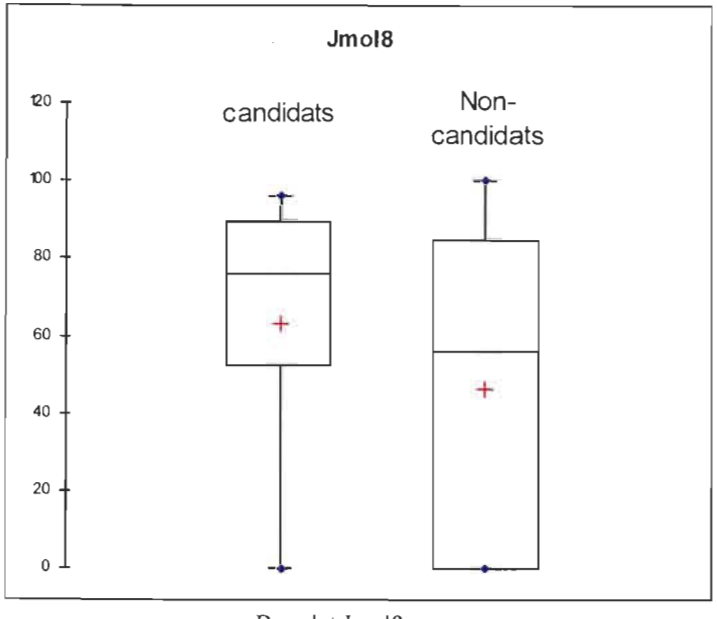
- **La métrique LCOM3**



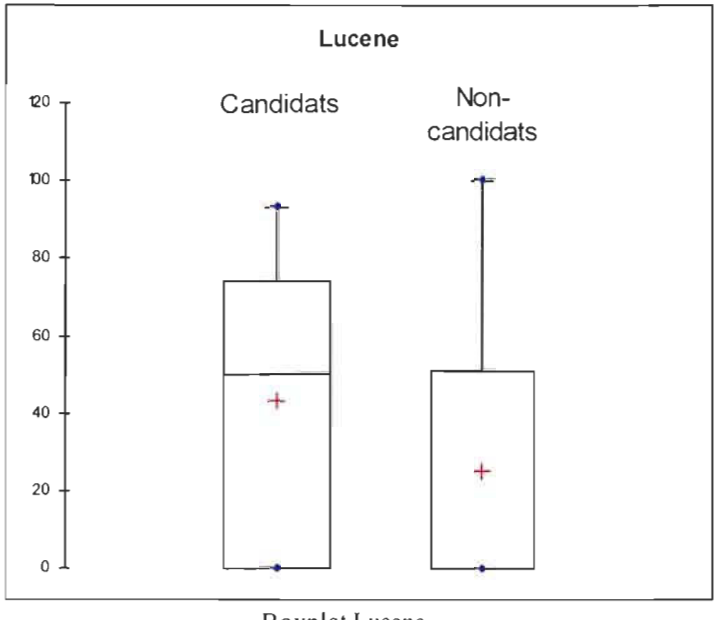
Boxplot JhotDraw.



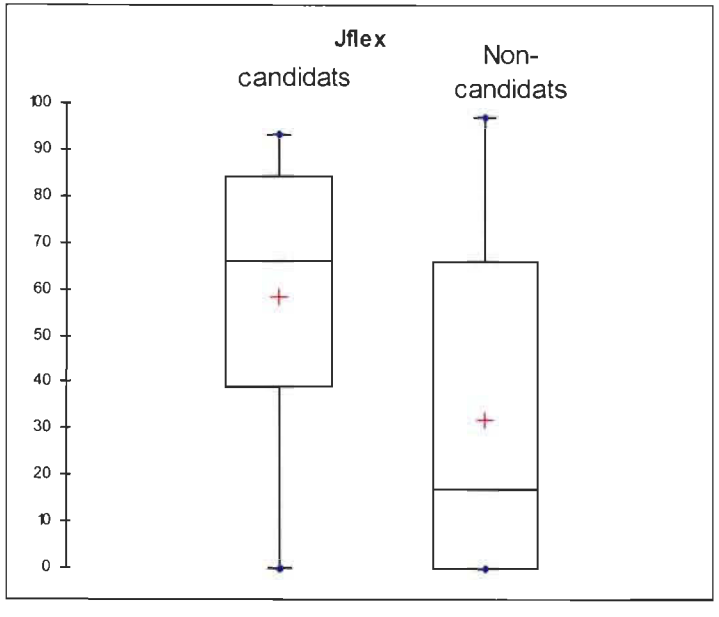
Boxplot FreeCS.



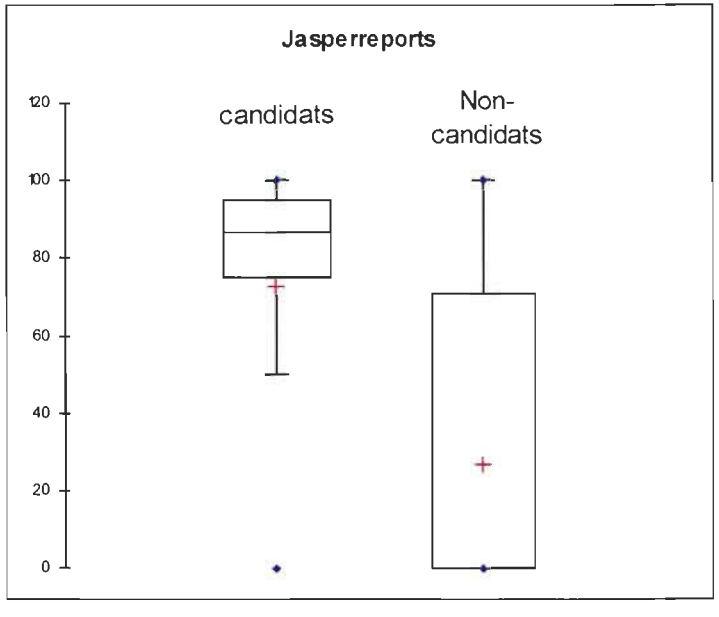
Boxplot Jmol8.



Boxplot Lucene.



Box plot Jflex.



Box plot JasperReports.

Figure 18: Box plots LCOM3 / Jhot., FreeCS, Jmol8, Lucene, Jflex, Jasper.

Avec des moyennes de 153.46, 752.75 et 387.68 pour les classes contenant du code correspondant à des préoccupations transverses (classes candidates) respectivement pour JhotDraw, FreeCS et JasperReports contre 15.82, 25.25 et 125.85 pour les classes ne présentant pas de disparité dans le code. Les tableaux VII, VIII, IX et X présentent des valeurs qui montrent que les classes ayant une faible cohésion (métrique permettant de calculer le manque de cohésion) ont une probabilité plus élevée de subir une restructuration (refactoring). Ceci se confirme par les autres caractéristiques comme la médiane, le premier et le troisième quartile pour chacune des métriques.

Tableau VII Statistiques descriptives JhotDraw.

		Minimum	Maximum	1er Quartile	Médiane	3ème Quartile	Moyenne
LDCd	candidats	0.0	1617.0	7.0	52.0	290.0	**208.4**
	Non-cand	0.0	620.0	0.0	2.0	9.0	15.6
LDCi	candidats	0.0	1614.0	7.0	52.0	282.0	**205.4**
	Non-cand	0.0	618.0	0.0	2.0	8.0	15.3
L-TCC	candidats	0.0	1627.0	8.0	53.0	297.5	**210.5**
	Non-cand	0.0	621.0	0.0	2.0	10.0	16.7
L-LCC	candidats	0.0	1624.0	7.0	53.0	293.0	**208. 596**
	Non-cand	0.0	621.0	0.0	2.0	9.0	16.5
LDCde	candidats	0.0	1575.0	7.0	51.0	278.0	**199.9**
	Non-cand	0.0	550.0	0.0	2.0	8.0	13.5
LDCie	candidats	0.0	1516.0	7.0	51.0	260.0	**190. 596**
	Non-cand	0.0	512.0	0.0	2.0	6.0	12.6
LCOM1	candidat	0.0	940.0	0.0	28.5	181.8	**144.4**
	N-candidat	0.0	391.0	0.0	0.0	3.0	9.9
LCOM2	candidats	0.0	100.0	56.0	83.0	93.0	**65.6**
	N-cand	0.0	100.0	0.0	0.0	67.0	30.5
LCOM3	candidats	0.0	100.0	0.0	50.0	79.3	**40.0**
	N-cand	0.0	100.0	0.0	0.0	0.0	11.8

Tableau VIII Statistiques descriptives JasperReports.

		Minimum	Maximum	1er Quartile	Médiane	3ème Quartile	Moyenne
LDCd	candidats	0	13833	10	28	138	**529.20**
	non-cand	0	14611	0	0	6	92.96
LDCi	candidats	0	13832	10	28	138	**525.30**
	non-cand	0	13832	10	28	138	525.30
L-TCC	candidat	0	13833	10	29	138	**533.20**
	non-cand	0	14611	0	1	6	94.68
L-LCC	candidat	0	13832	10	29	138	**530. 325**
	non-cand	0	14575	0	0	6	94.17
LDCde	candidat	0	13556	10	28	137	**516.08**
	non-cand	0	14254	0	0	5	89.69
LDCie	candidat	0	13466	10	28	130	**491.42**
	non-cand	0	13103	0	0	5	84.63
LCOM1	candidat	0	5121	9.25	34	153	**354.09**
	non-cand	0	13970	0	0	10	158.74
LCOM2	candidat	0	100	74	82	91	**79.45**
	non-cand	0	100	0	50	83	45.22
LCOM3	candidat	0	100	75	86.5	95	**72.72**
	non-cand	0	100	0	0	71	26.65

Tableau IX Statistiques descriptives FreeCS.

		Minimum	Maximum	1er Quartile	Médiane	3ème Quartile	Moyenne
LDCd	candidats	38	3963	100.25	351.5	1262	**1047.67**
	non-cand	0	375	1	1	9	20.14
LDCi	candidats	37	3955	100	350	1229.25	**1038. 500**
	non-cand	0	375	1	1	9	19.81
L-TCC	candidats	39	3963	100.25	352	1263	**1048.17**
	non-cand	0	375	1	1	9	20.20
L-LCC	candidats	39	3955	100	350.5	1232.5	**1039.67**
	non-cand	0	375	1	1	9	19.87
LDCde	candidats	34	3811	88.75	328.5	1168.25	**993.17**
	non-cand	0	353	1	1	7	19.22
LDCie	candidats	30	3659	88.25	316.5	1049.5	**938.00**
	non-cand	0	323	1	1	7	18.28
LCOM1	candidats	0	3514	47.5	105	935	**797.71**
	non-cand	0	282	2.25	3	6	17.38
LCOM2	candidats	0	96	83	92	94.5	**77.57**
	non-cand	0	100	57	83	83	66.68
LCOM3	candidat	0	100	86	94	97	**80.00**
	non-cand	0	100	0	0	66	25.63

Tableau X Statistiques descriptives PMD.

		Minimum	Maximum	1er Quartile	Médiane	3ème Quartile	Moyenne
LDCd	candidats	0	545	0	9	70	**69.24**
	non-cand	0	193	0	0	1	4.42
LDCi	candidats	0	490	0	9	70	**63.65**
	non-cand	0	193	0	0	1	4.42
L-TCC	candidats	0	655	0	9	99	**88.47**
	non-cand	0	193	0	0	1	4.44
L-LCC	candidats	0	655	0	9	99	**84. 353**
	non-cand	0	193	0	0	1	4.44
LDCde	candidats	0	545	0	9	70	**69.06**
	non-cand	0	183	0	0	1	4.31
LDCie	candidats	0	490	0	9	70	**63.47**
	non-cand	0	174	0	0	1	4.18
LCOM1	candidats	0	119700	0	0	3	**4464.22**
	non-cand	0	2346	0	0	6	28.73
LCOM2	candidat	0	95	0	0	62.5	**28.48**
	non-cand	0	100	0	0	61.5	28.21
LCOM3	candidats	0	100	0	0	53	**24.96**
	non-cand	0	100	0	0	6.25	18.24

V-5.2. Régression logistique

Les tableaux XI, XII, et XIII nous donnent les paramètres normalisés du modèle logistique de l'ensemble des métriques sélectionnées pour les systèmes JhotDraw, FreeCS, et JasperReports. Sont affichés pour ces différentes métriques, l'estimation du paramètre β, l'écart-type correspondant, le Khi2 de Wald, et les intervalles ≪ profile likelihood ≫.

Soit l'hypothèse H0 (H0 : Y = p0) : correspond au modèle indépendant qui donne la probabilité p0 (p0= 0.011 pour FreeCS ; 0.150 pour JasperReports ; 0.159 pour jHotDraw) pour les variables explicatives LDCd, LDCi, L-TCC, L-LCC, LDCde, et LDCie; et une probabilité qui oscillent entre 0.1 et 0.7 pour les variables explicatives LCOM1, LCOM2, LCOM3.

Nous cherchons à vérifier si notre modèle de prédiction est significativement plus performant que le modèle indépendant de l'hypothèse H0. Pour cela, nous effectuons le test de Wald suivant une loi du Khi2 à un degré de liberté.

Tableau XI : Coefficients normalisés des différentes métriques (Jhotdraw).

JhotDraw	β	Ecart-type	Khi² de Wald	Pr > Khi²
LDCd	0.770	0.170	20.391	< 0.0001
LDCi	0.774	0.172	20.224	< 0.0001
L-TCC	0.737	0.159	21.52	< 0.0001
L-LCC	0.737	0.159	21.529	< 0.0001
LDCde	0.872	0.191	20.75	< 0.0001
LDCie	0.903	0.197	20.992	< 0.0001
LCOM 1	0.801	0.185	18.783	< 0.0001
LCOM 2	0.801	0.185	18.783	< 0.0001
LCOM 3	0.41	0.076	28.939	< 0.0001

Tableau XII : Coefficients normalisés des différentes métriques (FreeCS).

FreeCS	β	Ecart-type	Khi² de Wald	Pr > Khi²
LDCd	2.241	0.825	7.381	0.007
LDCi	2.26	0.838	7.282	0.007
L-TCC	2.243	0.826	7.374	0.007
L-LCC	2.266	0,840	7,279	0.007
LDCde	2.255	0,822	7,520	0.006
LDCie	2.313	0,850	7.41	0.006
LCOM1	2.381	1,087	4.798	0.028
LCOM2	1.573	0,626	6.303	0.012
LCOM3	2.258	1,156	3.817	0.051

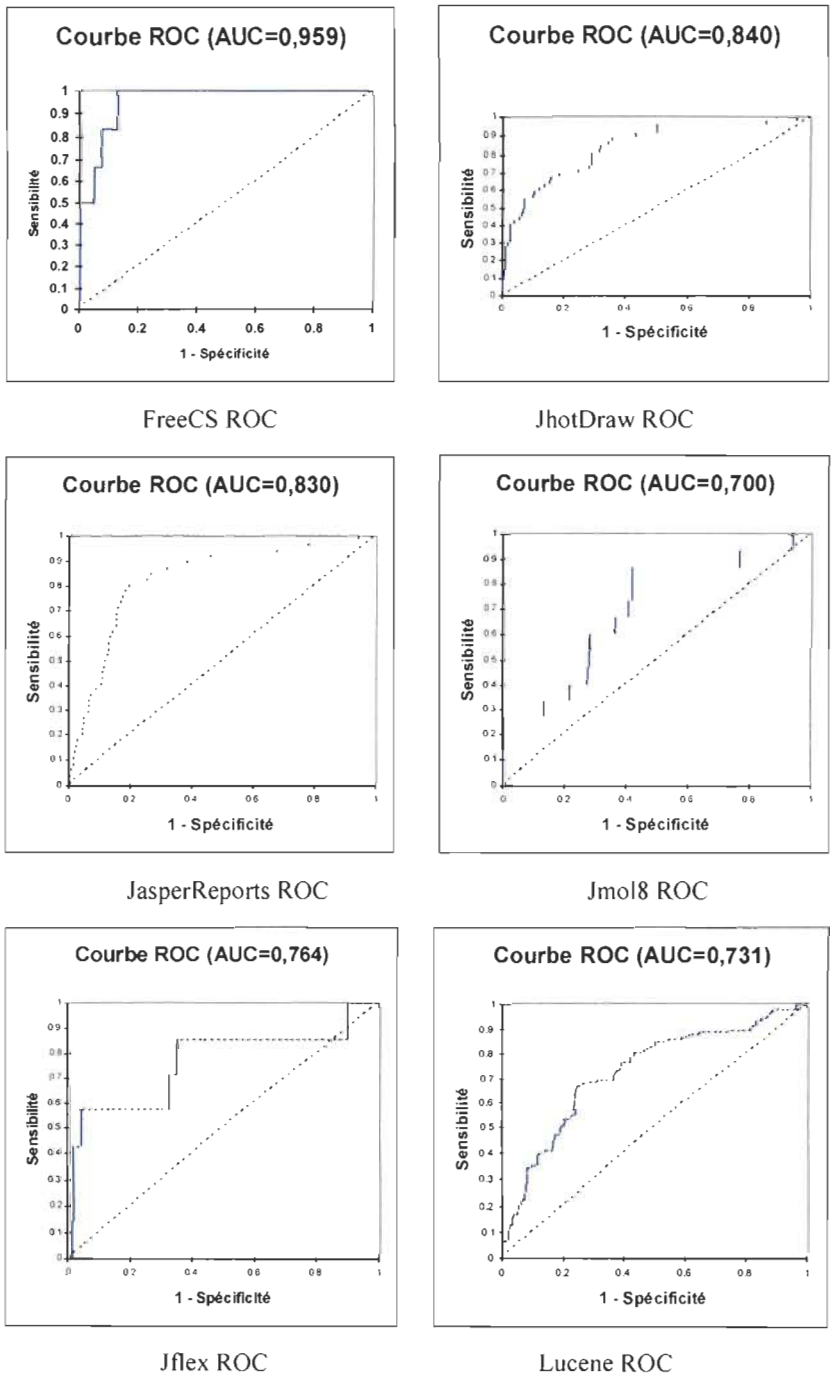
Tableau XIII : Coefficients normalisés des différentes métriques (JasperReports).

JasperReports	β	Ecart-type	Khi² de Wald	Pr > Khi²
LDCd	0.166	0.044	13.933	0
LDCi	0.165	0.044	13.834	0
L-TCC	0.166	0.044	14.055	0
L-LCC	0.166	0.044	13.99	0
LDCde	0.166	0.044	13.92	0
LDCie	0.168	0.045	13.728	0
LCOM1	0.026	0.054	0.233	0.63
LCOM2	0.315	0.081	14.991	0
LCOM3	0.139	0.069	4.077	0.043

La probabilité que l'hypothèse H0 (le manque de cohésion ne serait pas en rapport avec la présence de préoccupations transversales) du modèle indépendant se réalise est très faible. Pour le projet JHOTDRAW, par exemple, cette probabilité est (<0.001) pour toutes les métriques sélectionnées; pour FreeCS et JasperReports, elle est aussi négligeable, avoisinant zéro, surtout pour les métriques qui évaluent la cohésion avec des critères plus étendus (LDCde et LDCie), ce qui nous emmène à rejeter cette hypothèse (H0). Nous rappelons que ces métriques, grâce aux critères de cohésion complémentaires sur lesquels elles se fondent, capturent plus d'information, relativement aux paires de méthodes connectées, que les autres métriques telles que LCOM, L-TCC, et L-LCC. Nous pouvons donc dire que notre modèle de prédiction est plus performant que le modèle aléatoire. Le coefficient beta (normalisé) est largement au-dessus de zéro, ce qui démontre l'apport non négligeable en information de la cohésion dans l'identification des parties du programme contenant du code pouvant correspondre à des préoccupations transverses.

Les figures 19 à 27 représentent les courbes ROC des différents projets expérimentés. Ces courbes se démarquent bien de la première bissectrice. Elles nous montrent l'évolution de la spécificité et la sensibilité en fonction du point de séparation (cutoff).

- **La métrique LDCd**



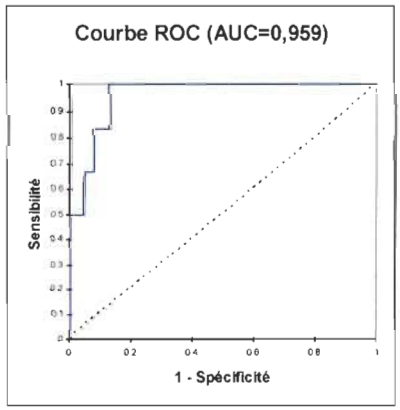
FreeCS ROC | JhotDraw ROC

JasperReports ROC | Jmol8 ROC

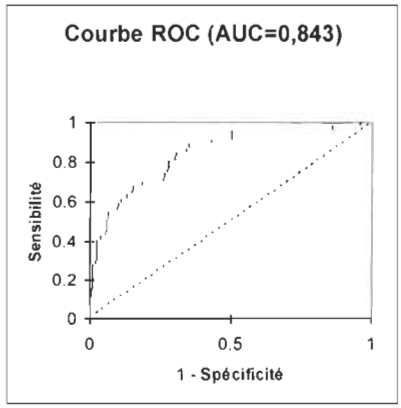
Jflex ROC | Lucene ROC

Figure 19: Courbe ROC LDCd /Jhot., Jflex, Lucene, FreeCS, Jmol8, Jasper.

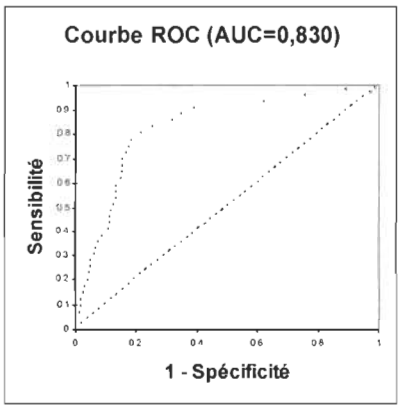
- **La métrique LDCi**



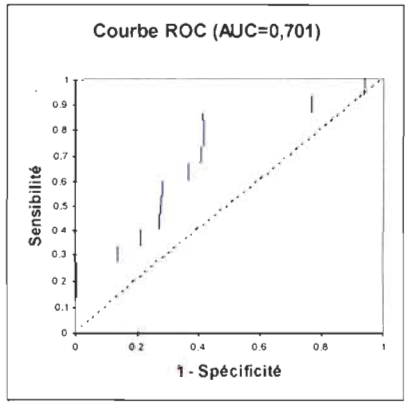
FreeCS ROC



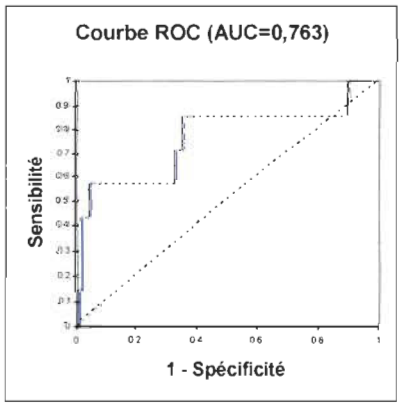
JhotDraw ROC



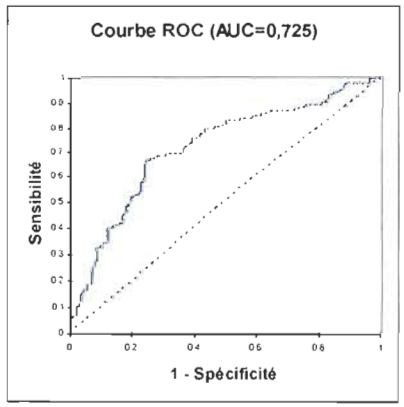
JasperReports ROC



Jmol8 ROC



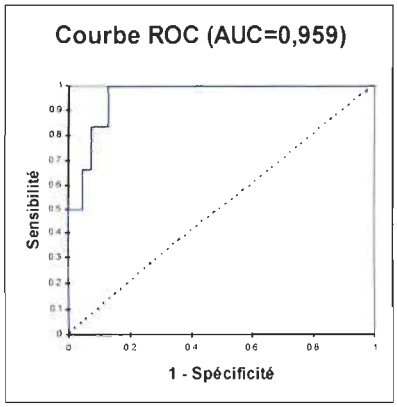
Jflex ROC



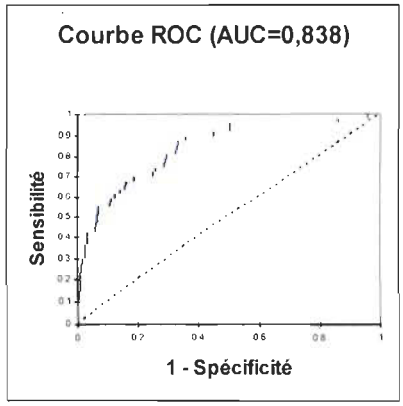
Lucene ROC

Figure 20: Courbe ROC LDCi / FreeCS,Jhot., Jasper., Jmol8, Jflex, Lucene.

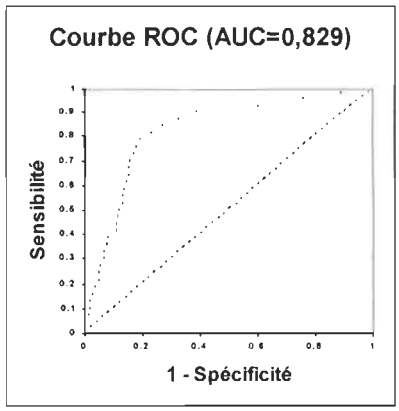
- **La métrique L-TCC**



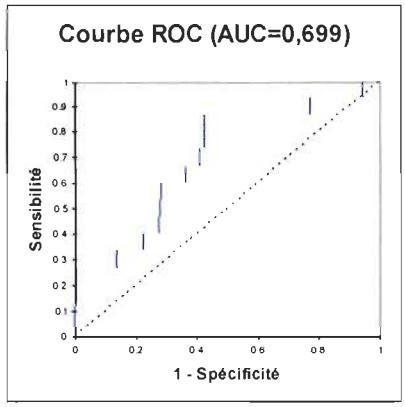
FreeCS ROC



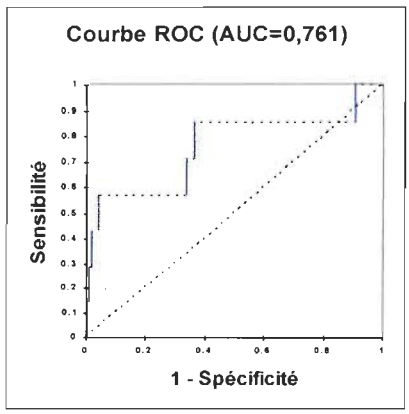
JhotDraw ROC



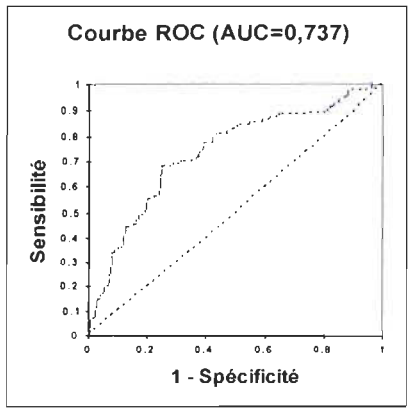
JasperReports ROC



Jmol8 ROC



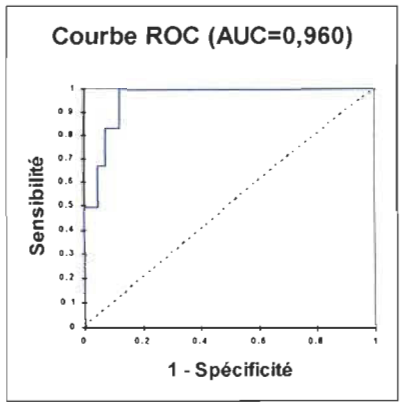
Jflex ROC



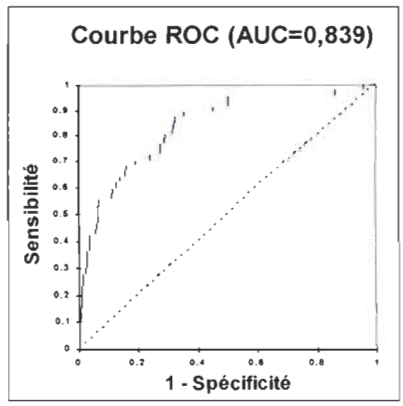
Lucene ROC

Figure 21: Courbe ROC L-TCC /FreeCS, Jhot., Jasper., Jmol8, Jflex, Lucene.

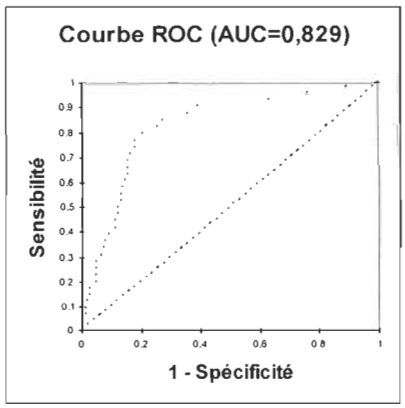
- **La métrique L-LCC**



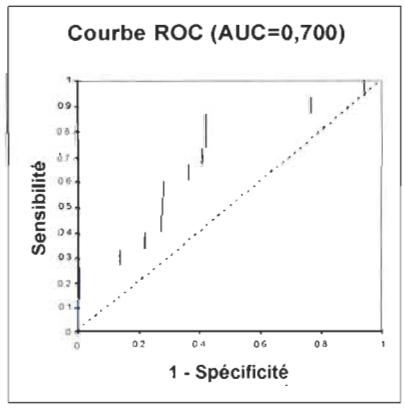
FreeCS ROC



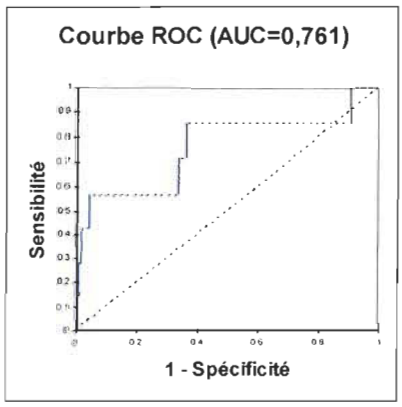
JhotDraw ROC



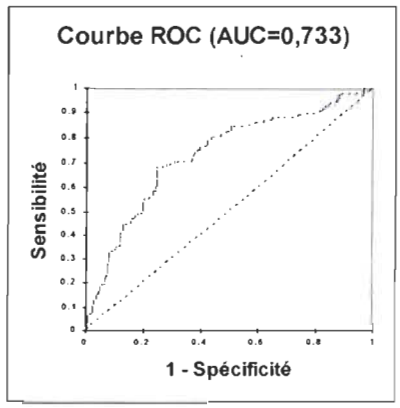
JasperReports ROC



Jmol8 ROC



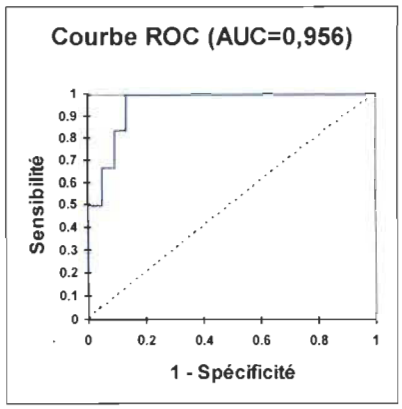
Jflex ROC



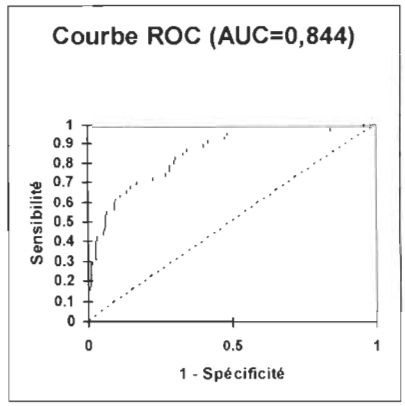
Lucene ROC

Figure 22: Courbe ROC L-LCC /FreeCS, Jhot., Jasper., Jmol8, Jflex, Lucene.

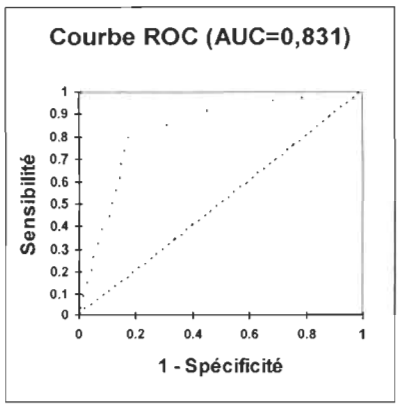
- **La métrique LDCde**



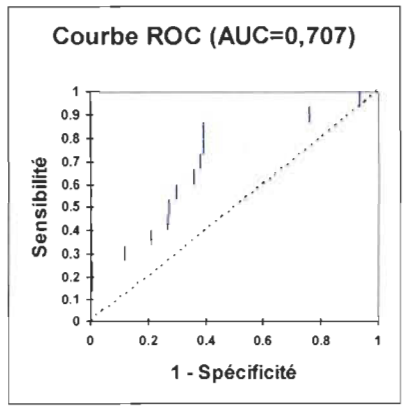
FreeCS ROC



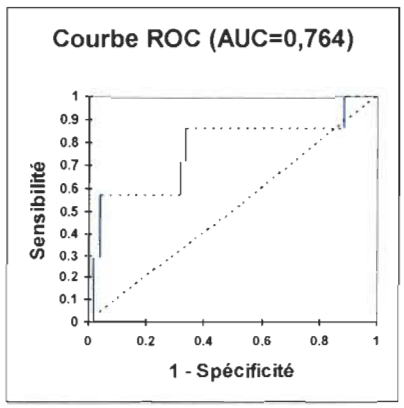
JhotDraw ROC



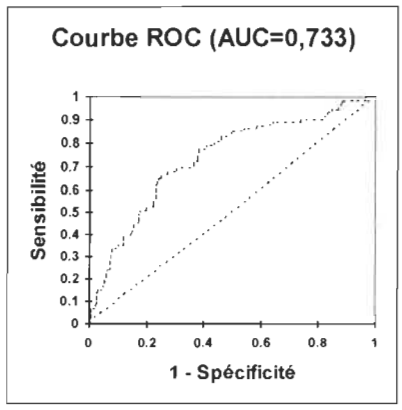
JasperReports ROC



Jmol8 ROC



Jflex ROC



Lucene ROC

Figure 23: Courbe ROC LDCde / FreeCS, Jhot., Jasper., Jmol8, Jflex, Lucene.

- **La métrique LDCie**

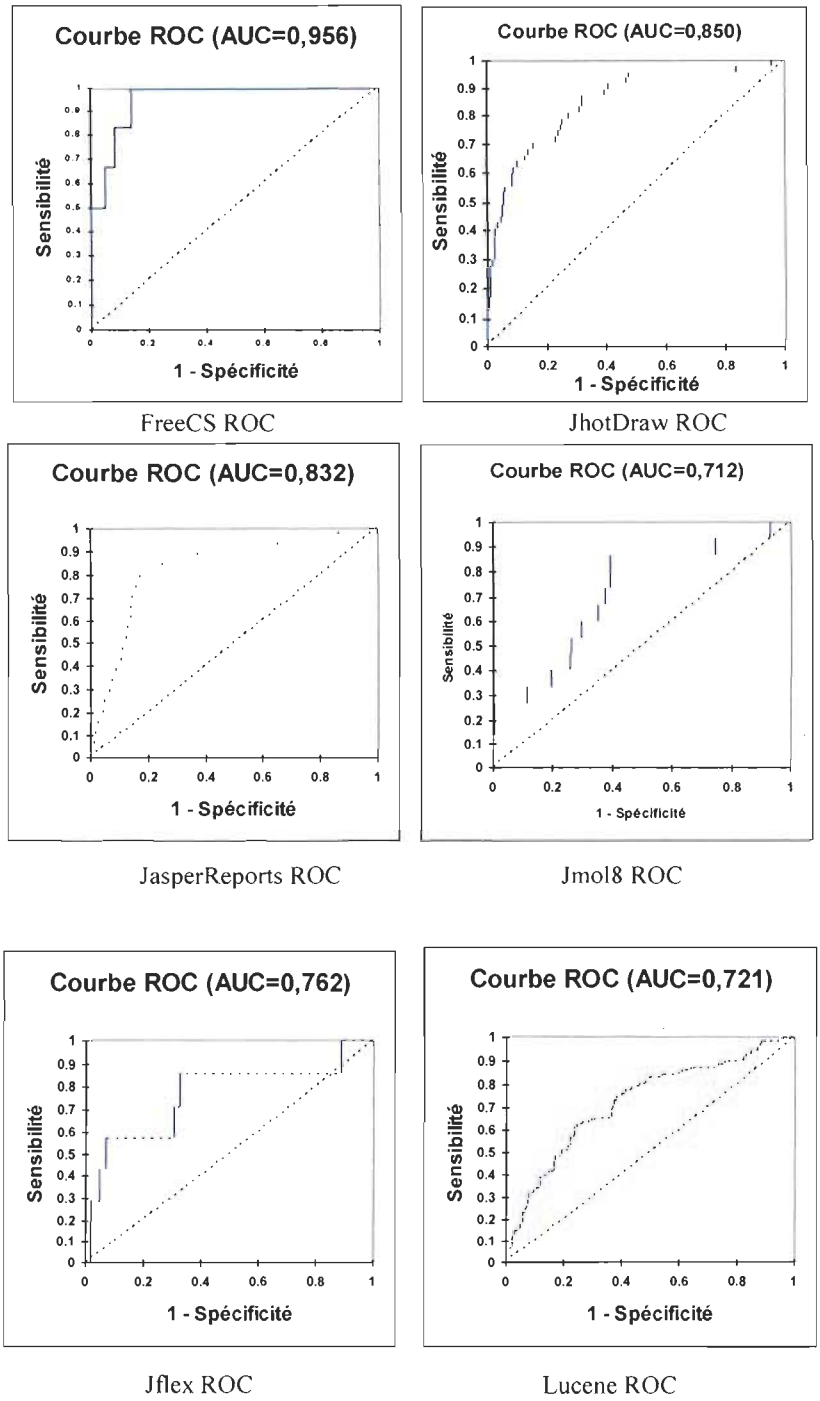
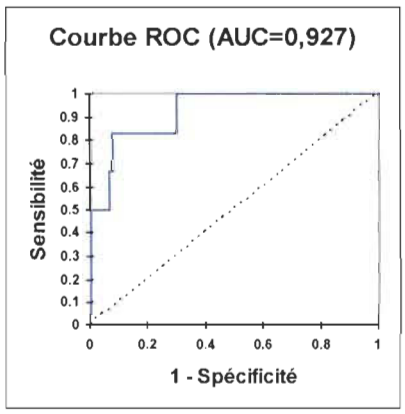
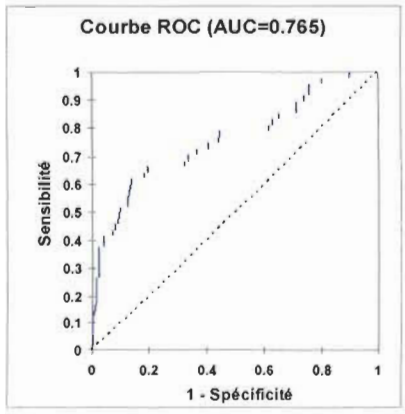


Figure 24: Courbe ROC LDCie / FreeCS, Jhot., Jasper.,Jmol8, Jflex, Lucene.

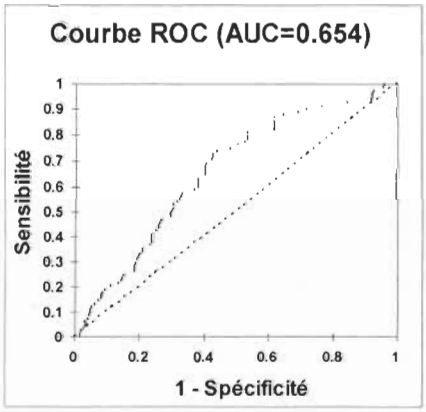
- **La métrique LCOM1**



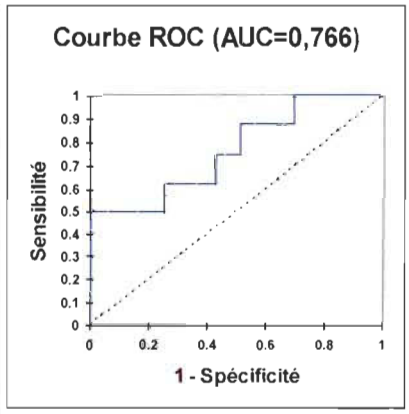
FreeCS ROC



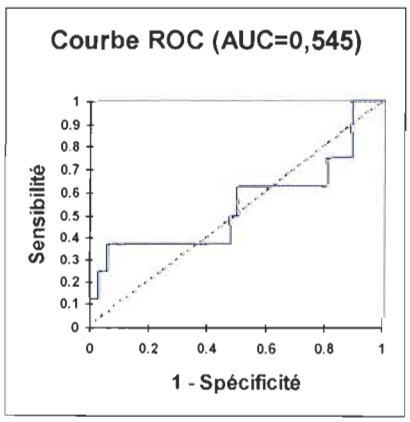
JhotDraw ROC



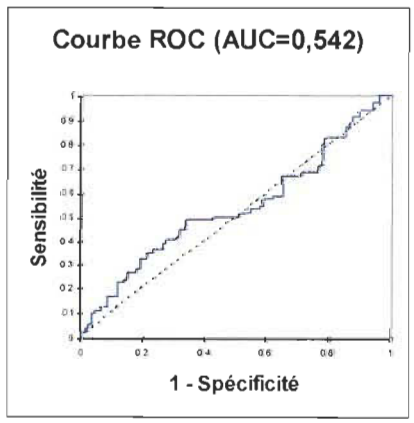
JasperReports ROC



Jmol8 ROC



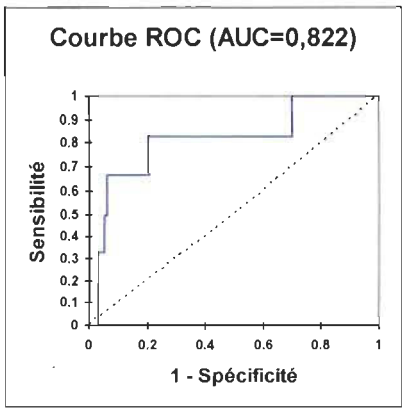
Jflex ROC



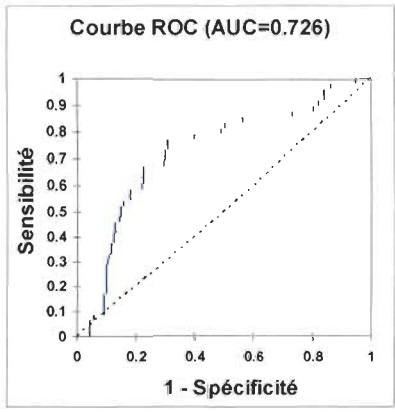
Lucene ROC

Figure 25: Courbe ROC LCOM1 / FreeCS, Jhot., Jmol8, Jasper., Jflex, Lucene.

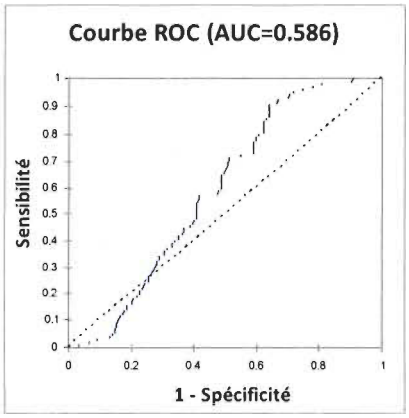
- **La métrique LCOM2**



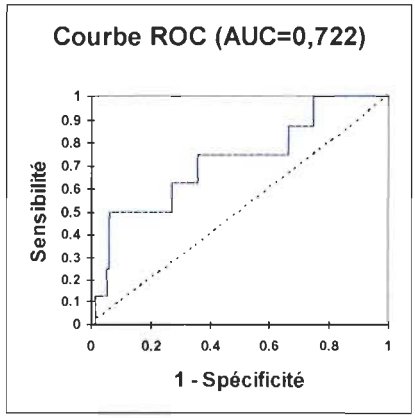
FreeCS ROC



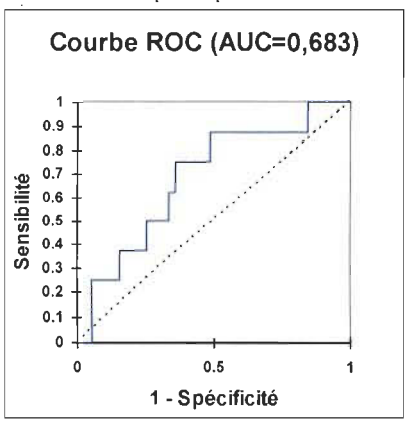
JhotDraw ROC



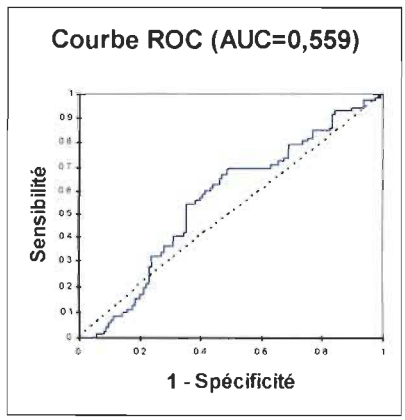
JasperReports ROC



Jmol8 ROC



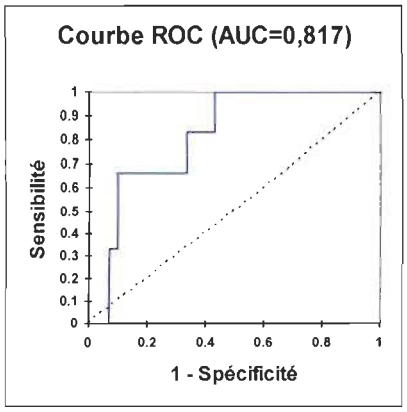
Jflex ROC



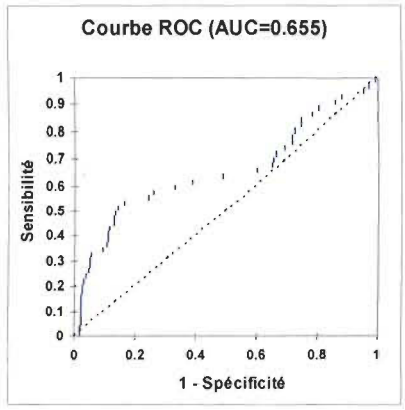
Lucene ROC

Figure 26: Courbe ROC LCOM2 / FreeCS, Jhot, Jasper., Jmol8, Jflex, Lucene.

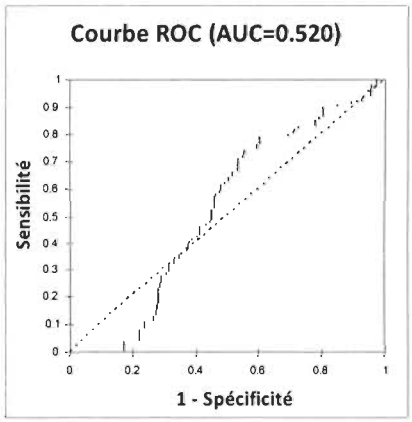
- **La métrique LCOM3**



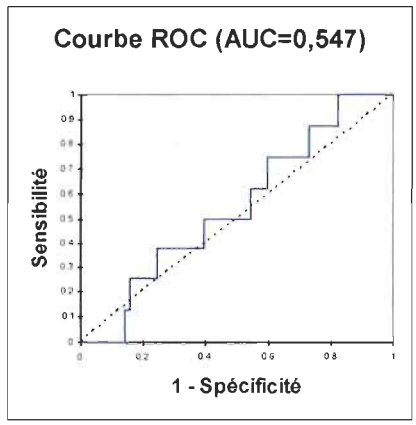
FreeCS ROC



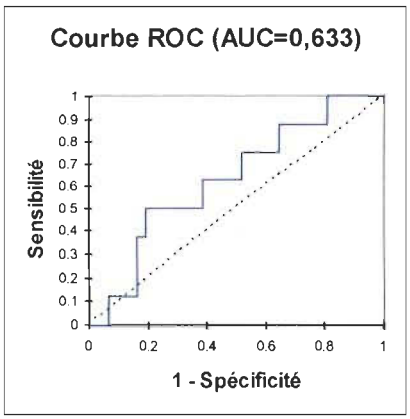
JhotDraw ROC



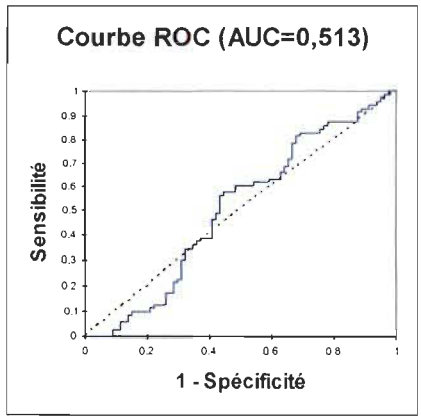
JasperReports ROC



Jmol8 ROC



Jflex ROC



Lucene ROC

Figure 27: Courbe ROC LCOM3 / FreeCS, Jhot., Jasper., Jmol8, Jflex, Lucene.

Au vu des résultats obtenus, en dehors des projets Lucene et FreeCS où les métriques respectives L-TCC et L-LCC présentent les meilleures capacités prédictives, dans l'ensemble de l'expérimentation, les meilleurs performances sont obtenues par les métriques LDCde et LDCie. Pour le cas de JhotDraw, qui est un projet assez expérimenté dans le domaine de l'aspect mining, les meilleures performances sont obtenues pour une spécificité égale à 0.5 et une sensibilité égale à 0.98 pour le cas de LDCde (inverse de DCde), et pour une spécificité égale à 0.45 et une sensibilité égale à 0.98 pour LDCie (inverse de DCie). L'AUC qui détermine la probabilité qu'un évènement positif ait une probabilité donnée par le modèle plus élevée qu'un évènement négatif et qui tient compte de la performance globale du modèle est égale à 0.844 pour LDCde et 0.85 pour LDCie (cas JhotDraw). Avec des valeurs de AUC supérieures à 0.7, on peut conclure que ce modèle de prédiction de classes contenant du code correspondant aux préoccupations transverses utilisant les métriques de cohésion est bien discriminant. Dans l'ensemble, les métrique LDCde, LDCie, LDCd, LDCi, L-TCC, et L-LCC sont de bons indicateurs de classes candidates. Les métriques LDCde, LDCie présentent les meilleures performances (ceci se justifie par leurs valeurs de AUC).

Tableau XIV: AUC pour les courbes ROC de toutes les métriques sélectionnées

	LDCd	LDCi	L-TCC	L-LCC	LDCde	LDCie	LCOM1	LCOM2	LCOM3
FreeCS	**0.959**	**0.959**	**0.959**	**0.96**	**0.956**	**0.956**	**0.927**	**0.822**	**0.817**
JhotDraw	**0.84**	**0.843**	**0.838**	**0.839**	**0.844**	**0.85**	**0.765**	**0.726**	**0.655**
JasperReports	**0.83**	**0.83**	**0.829**	**0.829**	**0.831**	**0.832**	**0.654**	**0.586**	**0.52**
PMD	**0.771**	**0.771**	**0.77**	**0.77**	**0.776**	**0.777**	**0.524**	**0.514**	**0.571**
Jflex	**0.764**	**0.763**	**0.761**	**0.761**	**0.764**	**0.762**	**0.545**	**0.683**	**0.633**
Lucene	**0.731**	**0.725**	**0.737**	**0.733**	**0.733**	**0.721**	**0.542**	**0.559**	**0.513**
Jmol8	**0.7**	**0.701**	**0.699**	**0.7**	**0.707**	**0.712**	**0.766**	**0.722**	**0.547**
Moyenne des corrélations	0.799	0.799	0.799	0.799	**0.802**	**0.801**	0.675	0.659	0.608

Le tableau XIV représente l'AUC pour les courbes ROC des différentes métriques sélectionnées pour l'ensemble des projets expérimentés. Dans l'ensemble, les résultats sont excellents pour la quasi-totalité des métriques. Les meilleures performances sont obtenues par les métriques LDCde et LDCie avec une moyenne de AUC respective de 0.802 et 0.801. Ceci confirme que ces métriques sont plus performantes que les autres métriques de cohésion et valident aussi le fait quelles capturent plus d'informations relatives à la cohésion des classes que les autres métriques. Ceci est du essentiellement aux critères de cohésion complémentaires sur lesquelles ces métriques (LDCde et LDCie) se basent. On remarque effectivement qu'elles sont les meilleurs indicateurs pour presque tous les projets expérimentés (JhotDraw, JasperReports, PMD, Jflex, et Jmol8).

Chapitre VI

Conclusions

Face à la complexité croissante des exigences, le développement des logiciels nécessite de plus en plus des techniques adéquates. La programmation orientée objet, qui a été largement utilisée dans l'industrie des logiciels, éprouve des difficultés à maîtriser, à différents points de vue, certains éléments de code dupliqués et dispersés dans les programmes. Ces « bouts » de code correspondent aux préoccupations transverses. La programmation orientée aspect, un nouveau paradigme de programmation, apporte une solution à ce problème. Elle offre des constructions permettant de rassembler ces éléments de code dans des unités modulaires appelées aspects. Pour le faire, le travail préliminaire consiste d'abord à identifier et localiser le code correspondant aux préoccupations transverses dans un programme (Aspect Mining). Notre projet s'inscrit dans ce cadre. Plusieurs techniques d'aspect mining existent dans la littérature, avec pour certaines des outils qui les supportent. Cependant, la plupart de ces outils existent sous forme de prototype, ce qui réduit leur accessibilité.

L'objectif de notre travail était d'explorer une nouvelle piste de recherche qui soutient l'identification des préoccupations transverses en utilisant la cohésion (ou le manque de cohésion). La cohésion est un des attributs les plus importants des systèmes orientés objet. Elle renseigne sur la qualité de la conception des composants (classes) logiciels. L'idée était de voir dans le contexte de notre étude jusqu'à quel degré la cohésion (ou le manque de cohésion – disparité dans le code) pouvait être utilisée pour localiser les parties de code pouvant correspondre à des préoccupations transverses. Ceci permettrait d'ouvrir une nouvelle piste de recherche pour supporter l'aspect mining. Par ailleurs, l'objectif était aussi d'évaluer de façon comparative plusieurs modèles de cohésion de la littérature.

Notre travail s'est, en fait, effectué en deux temps. D'abord, dans un premier temps, en s'appuyant sur une technique existante et validée dans le domaine d'Aspect Mining (analyse Fan-in), nous avons fait l'analyse statique du code de plusieurs projets écrits en Java. Le but était de déceler les classes contenant des symptômes de code correspondant à des préoccupations transverses en utilisant une technique éprouvée. Ensuite, nous avons évalué la cohésion des classes des différents projets sélectionnés pour notre expérimentation. Après la collecte et l'analyse des différentes données, nous sommes passés à la dernière phase de notre étude. Cette phase a porté sur l'évaluation des corrélations entre la cohésion et les résultats donnés par l'outil d'Aspect Mining utilisé. Comme pour toute étude de corrélation, nous avons besoin de visualiser les tendances de nos résultats. Alors, nous avons utilisé les courbes uni-variées pour visualiser les tendances. Grâces aux caractéristiques affichées par ces courbes, dont la moyenne des observations, nous avons pu constater que les classes candidates (classes contenant du code correspondant à des préoccupations transverses) ont une moyenne plus élevée (métriques de manque de cohésion) que les classes qui ne le sont pas. Ceci est confirmé par les autres caractéristiques des observations qu'affiche la courbe dont la médiane, le premier et le troisième quartile. L'analyse des données effectuée par la régression logistique, de part les résultats obtenus, nous permet de conclure sans risque de nous tromper que la cohésion peut être utilisée pour identifier les classes contenant du code disparate pouvant correspondre à des préoccupations transverses. Par ailleurs, l'évaluation comparative des métriques à permis de mettre en relief leurs performances respectives, et de montrer les métriques qui, dans la plupart des projets analysés, présentaient des corrélations très élevées. C'est le cas des métriques LDCde et LDCie. Ceci confirme aussi que ces métriques capturent plus d'informations relatives à la cohésion que les autres métriques évaluées.

Ce travail permet d'ouvrir de nouvelles pistes de recherche dérivées de notre méthodologie d'identification des préoccupations transverses qui pourraient aussi être utilisées pour supporter le refactoring aspect.

Chapitre VII

Références bibliographiques

[Aggarwal 06] K.K. Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra, "Empirical Study of Object-Oriented Metrics", in Journal of Object Technology, vol. 5. no. 8, November-December 2006 , pp. 149-173.

[Apel 07] Sven Apel, Christian Kästner, Thomas Leich, Gunter Saake "Aspect Refinement – Unifying AOP and Stepwise Refinement", in Journal of Object Technology, vol.6, no. 9, Special Issue. TOOLS EUROPE 2007, October 2007, pp. 13–33

[Badri 95] L. Badri, M. Badri et S. Ferdenache. "Towards Quality Control Metrics for Object-Oriented Systems Analysis". TOOLS (Technology of Object-Oriented Languages and Systems) Europe'95, Versailles, France, Prentice-Hall, March 1995.

[Badri 04] L. Badri & M. Badri: "A propasal of a New Class cohesion Criterion", Special issue (best papers) of Tools.USA 2003, April 2004.

[Badri 08] L. Badri, M. Badri et Alioune Gueye: "Revisiting Class Cohesion: An empirical investigation on several systems", in Journal of Object Technology (JOT), Juillet–Août 2008.

[Baltus 02] J. Baltus: "La Programmation Orientée Aspect et AspectJ: Présentation et Application dans un Système Distribué". Facultés Universitaires Notre-Dame de la Paix Namur, Belgique, 2002.

[Basili 96] V. Basili & L. Briand: "How reuse influences productivity in object-oriented. Systems", in Communications of the ACM, October 1996.

[Bieman 95] J.M. Bieman & B.K. Kang. "Cohesion and reuse in an object-oriented system". Proceedings of the Symposium on Software Reusability (SSR'95), Seattle, WA, pp. 259-262, April 1995.

[Breu 03] Silvia, Breu. and J. Krinke, "Aspect mining using dynamic analysis". In GI-Software technik-Trends, Mitteilungen der Gesellschaft für Informatik. Bad Honnef, Germany, 23: 21-22, 2003.

[Bruntink 04] Magiel Bruntink, A.V. Deursen, R.V. Engelen & T.Tourwe, 2004. "An evaluation of clone detection techniques for identifying crosscutting concerns". In Proceedings of the IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society Press, 2004.

[Borland 08] Borland Together 2008 Service Park1. Copyright 1998-2008 Borland Software Corporation. Visit Borland at http://www.borland.com

[Chidamber 91] Chidamber et Kemerer. "Towards a Metrics Suite for Object-Oriented Design". Object-Oriented Programming Vol. 26, No.10, pp. 197-211, October 1991.

[Chidamber 94] Chidamber et Kamerer, "A metrics Suite for Object-Oriented Design". IEEE Transactions on Software Engineering,.20, no.6, pp. 476- 493, August 1994.

[Éclipse 08] Éclipse (3.4.1) de IBM, (c) Copyright Eclipse contributors and others 2000, 2008. All rights reserved. Visit http://www.eclipse.org/platform. http://en.wikipedia.org/wiki/Eclipse_(software)

[Hannemann 01] Hannemann et Kiczales. "Overcoming the Prevalent Decomposition of Legacy Code", Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering (ICSE), 2001.

[Henderson 96] B. Henderson-sellers, "Object-Oriented Metrics, Measures of Complexity". Prentice Hall, 1996

[Hitz 95] Hitz, M. & B. Montazeri. "Chidamber and Kemerer's Metrics Suite: a measurement theory perspective". IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 22(4), April 1996, pp. 267-271.

[Kiczales 97] G. Kiczales, J. Lamping, Anurag Mendhekar, C. Maeda, C. Videira Lopes, Jean-Marc Loingtier, J. Irwin: "Aspect-Oriented Programming", in Proceeding of the Conference on Object-Oriented Programming (ECOOP), Finland, 1997.

[Kiczales 01] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., & Griswold W.G. "An Overview of AspectJ", In Proceedings of the 15th European Conference on Object-Oriented Programming. Knudsen Ed. Lecture Notes in Computer Science, vol. 2072. Springer-Verlag, London, 327-353, 2001

[Larman 03] Craig Larman : « UML et les design patterns », Campus Presse 2003.

[Li 95] W. Li, S. Henry, D. Kafura & R. Schulman. "Measuring Object-Oriented Design». In Journal of Object-Oriented Programming, Vol. 8, No. 4, pages 48-55, July/August 1995.

[Liu 00] K. Liu, S. Zhou, H. Yang "Quality Metrics of Object Oriented Design for Software Development and Re-development". IEEE, 2000.

[Marin 04] M. Marin, L. Moonen, & A. v. Deursen, "Identifying aspects using fan-in analysis", In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE2004), pages 132–141. IEEE Computer Society, 2004 pages 132–141. IEEE Computer Society, 2004.

[Marin 08] Marius Marin. "User Guide: FINT-Tool support for Aspect mining". Septembre 2008. http://swerl.tudelft.nl/bin/view/AMR/FINT.

[Bounour 06] Bounour N., Ghoul S., Atil F., "A Comparative Classification of Aspect Mining Approaches", J. Computer Sci., 2 (4): 322-325, 2006.

[Robillard 02] Robillard, M.P. and G.C. Murphy, "Concern graphs: Finding and describing concerns using structural program dependencies". In ICSE, 2002.

[Shepherd 04] David Shepherd & Lori Pollock, "Automated Mining of Desirable Aspects", International Conference on Software Engineering Research and Practice (SERP 2004), June 2005.

[Sommerville 04] Ian Sommerville, "Software ingeneering" seventh edition British library p492, 2004.

[Tonella 04] Paolo Tonella & M. Ceccato, "Aspect mining through the formal concept analysis of execution traces". In Proceedings of the 11th Working Conference on Reverse Engineering, IEEE Computer Society, pp: 112-121, 2004.

[Tourwé 04] Tom Tourwé et Kim Mens "Mining aspectual views using formal concept analysis". In 4th IEEE Intl. Workshop on Source Code Analysis and Manipulation, pp: 97-106.

[Volder 02] Volder, "The jquery tool: A generic query-based code browser for eclipse". Presentation at Eclipse. BoF at OOPSLA 2002.

[XLSTAT 07] www.xlstat.com

[Yuan 99] J.J. Yuan, William G.G., & Y. Kato, 1999: "Aspect browser: Tool support for Managing Dispersed Aspects". Technical Report CS99-0640, Department of computer Science and Engineering, University of California, San Diego.

[Zhang 03] Charles Zhang, et H.-A. Jacobsen, "A Prism for Research in Software Modularization through Aspect Mining", Technical report, Middleware Systems Research Group, University of Toronto, 2003.