UNIVERSITÉ DU QUÉBEC

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

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
LOUIS HAMEL

DÉTECTION DES PRÉOCCUPATIONS TRANSVERSES AVEC APPRENTISSAGE AUTOMATISÉ

AOÛT 2022

Détection des préoccupations transverses avec apprentissage automatisé

Louis Hamel

RÉSUMÉ

Les préoccupations transverses sont un enjeu incontournable dans le développement d'un logiciel. La programmation orientée aspect a été proposée en 1997 pour permettre de modulariser ces préoccupations transverses autrement éparpillées au travers du logiciel. L'*aspect mining* est une activité qui consiste à chercher des préoccupations transverses présentes dans un logiciel pour pouvoir les remanier avec la programmation orientée aspect. Le couplage entrant est un des principaux indicateurs utilisés pour détecter des préoccupations transverses à partir du code source. Nous soupçonnons les préoccupations transverses d'avoir un impact sur la cohésion des classes. L'objectif de ce mémoire est de vérifier si adjoindre la cohésion au couplage entrant améliore la détection des préoccupations transverses. Nous explorons, entre autres, l'utilisation de l'apprentissage automatisé avec des métriques de couplage et de cohésion. Nous constatons que la contribution des métriques est variable d'un système à l'autre et que la cohésion peut améliorer la précision. Nous étudions aussi la capacité des métriques à détecter avec précision des éléments complémentaires. Il en ressort que plus d'une métrique de cohésion peuvent détecter des éléments complémentaires. Nous terminons en discutant de méthodes alternatives utilisées pour gérer les préoccupations transverses.

Mots-clés: Aspect Mining, Programmation orientée aspect, Apprentissage automatisé

Crosscutting concerns detection using machine learning

Louis Hamel

ABSTRACT

Crosscutting conserns is an important issue in the development of software. Aspect oriented programming was proposed in 1997 to allow modularization of those crosscutting concerns, which are otherwise scattered through the software architecture. Aspect mining is the activity of researching existing crosscutting concerns in an existing software in order to refactor them using aspect oriented programming. Fan-in analysis is a well known approach for finding crosscutting concerns from source code. We suspect crosscutting concerns to have an impact on cohesion. This research aims to find out if combining cohesion with fan-in can improve crosscutting concerns detection. We explore the use of machine learning with fan-in and cohesion metrics. We find out that metrics contribution differs between systems and that cohesion may improve precision. We also study the complementarity of metrics for detecting precisely crosscutting concerns. We find out that different cohesion metrics may detect complementary elements with high precision. We conclude this reseach with a disscussion about alternative ways of handling crosscutting concerns.

Keywords: Aspect Mining, Aspect Oriented Programming, Machine Learning

AVANT-PROPOS

Le professeur pointe le télescope vers les bonnes étoiles pour que l'étudiant imagine les lignes des constellations. Le chercheur, seul avec son télescope, cherche ses étoiles et imagine ses lignes.

REMERCIEMENTS

Je remercie dans un premier temps mon directeur de recherche, le professeur Mourad Badri, pour m'avoir initié à la recherche dans le domaine palpitant du génie logiciel et pour son précieux soutien au cours des dernières années. Je remercie aussi les professeurs qui m'ont enseigné au cours de mon parcours universitaire et qui m'ont transmis avec enthousiasme de précieuses connaissances. Je remercie aussi mes camarades avec qui j'ai pu approfondir des réflexions sur les connaissances enseignées.

Je remercie aussi le Conseil de recherche en sciences naturelles et en génie du Canada (CRSNG) et le Fond de recherche du Québec en science et technologies (FRQNT) pour leur soutien financier.

Je remercie aussi Pierre-Olivier Parisé, Nicolas-Félix Lacombe, Jean-François Pruneau, Keven Dubé et Katlyn Thibodeau pour les nombreuses discussions philosophiques et humaines.

Je remercie mes parents, Jean-Yves et Johanne ainsi que mon frère Élie pour leur soutien et leur aide. Je remercie spécialement Thérèse, dont la chaleur a marqué les débuts de ma mémoire à l'aube de ma vie mais au crépuscule de la sienne, et dont l'héritage fait encore vibrer cœurs et oreilles par-delà les générations.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

CHAPITRE 1

INTRODUCTION

La programmation orientée objet est un paradigme de programmation largement utilisé lors des dernières décennies. Ce paradigme permet d'élaborer une solution logicielle en modélisant un sous-ensemble de la logique du domaine métier par un ensemble de classes. Ces classes servent à créer des objets représentant, entre autres, des versions simplifiées d'entités du domaine métier. La programmation orientée objet est une partie du paradigme orienté objet, qui englobe non seulement la programmation mais aussi la modélisation et la conception du logiciel. L'une des forces de l'orienté objet est de pouvoir être aisément utilisé dans une grande variété de logiques métier.

La modularité est un élément important de la qualité d'un logiciel. Il est souhaitable pour un système de maximiser la cohésion de ses composants tout en minimisant leur couplage. La cohésion est définie comme « une mesure de l'étroitesse des liens et de la spécialisation des responsabilités d'un élément » (Larman *et al.*, 2005). La cohésion d'une classe lui permet d'être plus facile à comprendre, à réutiliser et à maintenir. Un couplage faible permet quant à lui de limiter l'impact des modifications sur un élément (Larman *et al.*, 2005). Les patrons de conception GRASP, dont font partie la forte cohésion et le faible couplage, ont pour but d'aider à déterminer à quels éléments affecter les responsabilités selon une conception orientée objet (Larman *et al.*, 2005).

La gestion des préoccupations transverse est en enjeu dans la programmation orientée objet. Les préoccupations transverses sont des besoins difficiles à isoler qui se retrouvent éparpillés parmi divers modules d'un logiciel (Marin, Deursen & Moonen, 2004). Elles transcendent le découpage principal du logiciel. Leur prise en charge est associée à de la duplication (Bruntink, Deursen, Engelen & Tourwe, 2005) et à de l'enchevêtrement de code (Kiczales *et al.*, 1997).

Les préoccupations transverses sont des responsabilités subordonnées aux responsabilités principales du système. La programmation orientée objet ou procédurale ne permet pas d'implémenter ces responsabilités entièrement dans une unité cohésive et faiblement couplée.

La journalisation est un exemple de préoccupation transverse. Elle consiste à générer des traces datées (avec l'heure) d'évènements importants lors de l'exécution du programme. C'est un élément important pour enquêter sur le fonctionnement du programme, notamment en cas de bogue. La prise en charge de la journalisation implique d'ajouter des lignes de code aux endroits dans le code où l'on veut qu'une trace soit générée. Bien que minime soit la quantité de lignes de code à subordonner aux fonctionnalités principales d'une classe, il reste nécessaire d'ajouter des lignes de code à tous les endroits où une trace doit être générée. Et les classes concernées seront tout de même couplées à un module de journalisation.

La programmation procédurale et orientée objet échouent à capturer et à isoler les préoccupations transverses (Kersten, Matwin, Noronha & Kersten, 2000). Certains patrons de conception GoF amènent des structures transverses (Hannemann & Kiczales, 2002).

La programmation orientée aspect (POA) est un paradigme pouvant être greffé à la programmation orientée objet, mais aussi à la programmation procédurale. Elle a été proposée pour améliorer la prise en charge des préoccupations transverses. La programmation orientée aspect permet d'encapsuler des préoccupations transverses dans des modules appelés aspects (Kiczales *et al.*, 1997). Dans un aspect, on peut définir des instructions dans des *advices*. Les *advices* pourront être tissés à des points de coupure, un ensemble de points de jonction dans le système principal auquel l'aspect sera relié. Ces instructions seront tissées aux points de coupures par un tisseur d'aspect, soit lors de la compilation, soit lors de l'exécution du programme. AspectJ est une extension orientée aspect de Java.

Le *remaniement aspect* est l'activité consistant à transformer un système en un système orienté aspect. Il s'agit de trouver les préoccupations transverses déjà développées dans le code et ensuite de les remanier en aspects. L'étape consistant à trouver ces préoccupations transverses est appelée aspect mining. Il y a plusieurs techniques d'aspect mining. Certaines analysent le code source, d'autres se concentrent sur la définition des spécifications du système. Dans ce mémoire, on s'intéresse à l'aspect mining utilisant des métriques des classes du code source. Il s'agit d'analyse statique du code. L'outil FINT a été proposé en 2006 (Marin, Moonen & Deursen,

2006) afin d'assister les programmeurs dans l'analyse manuelle du code. L'outil calcule certaines métriques, dont le couplage entrant, et permet de filtrer les éléments du code selon les valeurs de ces métriques.

1.1 Objectif

Les travaux présentés dans ce mémoire ont pour objectif de proposer une approche d'aspect mining basée sur des métriques de classe. Les objectifs de l'aspect mining sont de :

1. Détecter un maximum de classes investies de préoccupations transverses ;
2. Détecter les classes investies de préoccupations transverses avec un maximum de précision.

La nouvelle approche doit pouvoir prédire pour chaque classe si elle est porteuse ou non d'au moins une préoccupation transverse. Les deux objectifs sont équivalents à minimiser les faux négatifs (premier objectif) et minimiser les faux positifs (second objectif).

1.2 Questions de recherche

On utilise l'analyse du couplage entrant proposé par l'outil FINT comme approche de contrôle. Notre question de recherche principale est la suivante :

QR : Est-ce que combiner le couplage entrant à d'autres métriques de classe peut améliorer la détection des préoccupations transverses ?

Le choix des métriques, la façon de les combiner et l'amélioration de la détection sont trois aspects de cette question de recherche. Les sous-questions de recherche ci-dessous approfondissent ces aspects.

QR1 : Est-ce que combiner les métriques permet de détecter des éléments complémentaires aux éléments détectés par le couplage entrant ?

QR2 : Est-ce que combiner les métriques permet d'augmenter la précision de la détection ?

QR3 : Est-ce qu'un algorithme d'apprentissage supervisé peut améliorer la détection des préoccupations transverses ?

QR4 : Est-ce que l'utilisation de métriques de cohésion peut améliorer la détection des préoccupations transverses ?

Les deux premières questions de recherche concernent l'atteinte des objectifs de l'aspect mining. Les deux autres questions de recherches orientent vers les pistes explorées dans ce projet.

Le chapitre 2 présente des travaux connexes. Le chapitre 3 présente un travail antérieur explorant des métriques de cohésion. Le chapitre 4 explore l'utilisation de l'apprentissage supervisé pour combiner les métriques. Le chapitre 5 étudie la complémentarité des métriques et discute d'une heuristique pour en trouver de nouvelles. Le chapitre 6 présente une réflexion complémentaire à ce projet en discutant d'autres approches de prise en charge des préoccupations transverses.

CHAPITRE 2

TRAVAUX RELIÉS

Après l'arrivée de la programmation orientée aspect en 1997, la recherche sur l'aspect mining a connu un essor au cours des années 2000. Plusieurs approches ont été proposées et explorées.

Certaines techniques recherchent des correspondances textuelles (Griswold, Kato & Yuan, 2000), d'autres utilisent les traces d'exécution (Breu & Krinke, 2004; Tonella & Ceccato, 2004).

(Canfora & Cerulo, 2005) ont étudié l'évolution des préoccupations transverses dans le système JHotDraw au fil des versions. Ils ont remarqué que leur introduction et leur maintenance se font par des séries de modifications faites exclusivement pour ces préoccupations. (Breu & Zimmermann, 2006) ont proposé une approche d'aspect mining utilisant l'historique des versions.

(Marin, Moonen & Deursen, 2005) présentent une classification des préoccupations transverses. Ils présentent, pour chaque type de préoccupation, la manière habituelle de les prendre en charge dans un système orienté objet et les mécanismes proposés par AspectJ pour les prendre en charge.

(Ceccato *et al.*, 2006) étudient la combinaison de techniques d'aspect mining. Combiner différentes techniques complémentaires permettrait d'augmenter la couverture de détection.

FINT est un outil d'aspect mining reposant sur l'analyse du couplage entrant (Marin *et al.*, 2006). FINT propose trois approches. L'analyse du couplage entrant compte pour chaque méthode le nombre d'appels faits vers elle. L'analyse d'appels groupés détecte des groupes de méthodes ayant plusieurs appelants communs. La recherche de redirections cherche les classes dont les méthodes redirigent simplement l'exécution vers des méthodes d'une autre classe. Ces trois approches mettent en évidence des graines (appelées *seeds* en anglais), c'est-à-dire des éléments du code servant de points de départ pour enquêter sur une préoccupation.

Zhang et al. ont proposé une approche de détection des préoccupations transverses utilisant des marches aléatoires (Random Walks) (Zhang & Jacobsen, 2012). Le principe est de simuler des

marches d'exploration dans le système en suivant les dépendances d'utilisation entre les classes. Ces marches servent à calculer pour chaque classe une métrique d'utilisation et une métrique d'agrégation.

CHAPITRE 3

LA COHÉSION

Avant d'aborder directement la détection des préoccupations transverses, nous avons d'abord travaillé sur la recherche d'une métrique de cohésion. Le besoin pour cette recherche vient du fait qu'aucune métrique ne fait encore l'objet d'un consensus dans la communauté scientifique pour évaluer la cohésion. Le projet consistait à poursuivre l'exploration de l'utilisation de la classification hiérarchique ascendante, débutée par Lazhar Sadaoui (Sadaoui, Badri & Badri, 2012). Ses travaux présentent une métrique calculée à partir du résultat d'une classification hiérarchique ascendante (CHA).

Le principe d'une classification hiérarchique ascendante est d'agglomérer successivement les groupes d'éléments les plus similaires. Au départ, chaque élément est placé dans un groupe individuel. À chaque agglomération, le niveau de similarité entre les deux groupes fusionnés est noté. Les fusions sont effectuées itérativement jusqu'à ce qu'il n'y ait qu'un seul groupe agglomérant tous les éléments.

3.1 La métrique COHCL

La métrique proposée par Sadaoui et al. (Sadaoui *et al.*, 2012) est calculée comme suit. Pour une classe logicielle donnée, on construit une matrice entité-propriétés. Les entités sont les membres de la classe (attributs, méthodes) et les propriétés représentent leurs relations. Chaque membre de la classe est mis en relation avec les autres membres, selon que ce membre invoque ou non chacun des autres membres. Un indice de similarité, soit l'indice de Jaccard, est appliqué pour construire une matrice de dissimilarité entre les entités (dissimilarité = 1 − similarité). On applique une CHA sur la matrice de dissimilarité. La CHA produit un dendrogramme, lequel est tronqué à l'aide d'une fonction de troncature automatisée intégrée au logiciel XLSTAT. Cette fonction est présumément basée sur le principe d'entropie, mais n'est pas explicitée. À partir de la classification, un graphe de connexion est construit. Ce graphe présente les méthodes seulement, reliées à toutes les autres méthodes présentes dans la même classe de la CHA. La

métrique est calculée en divisant le nombre de connexions présentes par le nombre de connexions possibles.

N représente le nombre de nœuds, soit le nombre de méthodes de la classe, et NC représente le nombre de connexions dans le graphe de connexions.

$$COH_{CL} = \frac{2N_C}{N \cdot (N-1)} \tag{3.1}$$

Les travaux de (Sadaoui *et al.*, 2012) mentionnent aussi le nombre de composantes connexes (**NCC**) du graphe de connexion. NCC correspond au nombre de clusters (**NbCluster**) issus de la CHA.

3.2 Limites de COHCL

L'utilisation d'XLSTAT pour calculer la métrique pose un problème. La fonction de troncature du dendrogramme est opaque. La version de XLSTAT utilisée à l'époque n'indique pas comment le niveau de troncature optimal est calculé. On sait seulement que cette fonction est basée sur l'entropie. Cette opacité nuit à la démarche scientifique car cela rend les définitions des métriques COHcl et NbCluster opaques et dépendantes de XLSTAT.

3.3 La métrique COHCL2

Notre travail sur la cohésion a mené à une seconde métrique qui ne nécessite pas la fonction opaque de l'outil XLSTAT.

La première étape consiste à construire le graphe structurel de la classe. Les nœuds correspondent aux méthodes et aux attributs de la classe. Il y a une arête entre les nœuds de deux méthodes si au moins une des deux méthodes invoque l'autre. Il y a une arête entre le nœud d'une méthode et le nœud d'un attribut si la méthode utilise directement l'attribut, que ce soit en lecture ou en écriture. Les nœuds de deux attributs ne sont jamais reliés. On testera ensuite la connexité

du graphe en comptant le nombre de composantes connexes. Les composantes ayant un seul élément sont ignorées - il peut s'agir de constantes. La constitution de la matrice entité-propriété et le choix de l'indice de similarité différera selon qu'il y ait une seule ou plus d'une composante connexe comptée.

La matrice entité-propriété est construite à partir du graphe structurel. Les entités représentent les membres de la classe et les propriétés de ces entités sont leurs relations avec chacun des membres de la classe. Un attribut est toujours considéré en relation avec lui-même. Une méthode m_i est considérée en relation avec une méthode m_j si mi invoque m_j. Une méthode est donc considérée en relation avec elle-même seulement si elle est récursive. Pour chaque propriété d'une entité, la valeur est nulle s'il n'y a pas de relation. Un attribut a toujours la valeur 1 vis-à-vis de lui-même. Pour une entité représentant une méthode, les valeurs de ses propriétés représentent le nombre d'invocations de chaque membre. On utilise le cosinus comme indice de similarité. La méthode de troncature consiste à tronquer au niveau de similarité le plus faible mais non nul.

Le graphe de connexion est construit à partir des agglomérats résultants de la troncature. Les nœuds représentent les méthodes seulement. Les méthodes présentes dans un même cluster sont reliées. On relie aussi les méthodes qui s'appellent ou qui appellent un même membre. À partir de cette étape, le calcul du nombre de composantes connexes (NCC) et de COHCL2 est identique à la première approche.

Ces métriques sont présentées dans ce mémoire, car nous soupçonnons la cohésion d'être affectée par la présence de préoccupations transverses. D'un point de vue conceptuel, la présence d'une préoccupation transverse implique qu'une responsabilité supplémentaire est affectée à la classe. C'est donc une diminution de la cohésion (risque d'augmentation de la disparité). Toutefois, aucune métrique de cohésion ne fait consensus, car aucune ne capture tous les aspects conceptuels de la cohésion. Il n'y a donc pas de garantie qu'un manque de cohésion relié à une préoccupation transverse soit effectivement capturé par une métrique de cohésion.

3.4 Limites de COHCL2

La méthode de troncature du dendrogramme a toujours tendance à créer minimalement deux clusters, ce qui inhibe des liens dans le graphe de connexions. Ainsi, à l'exception de cas dégénérés, la métrique COHCL2 n'indiquera jamais une cohésion de 100%, même si la classe est conceptuellement très cohésive.

Cette nouvelle approche, ainsi que celle proposée originellement par Sadaoui, est aussi limitée à étudier les liens structurels entre les membres d'une même classe. Si une méthode appelle un membre qui est défini dans une autre classe, ce lien structurel n'est pas considéré dans la définition de la métrique.

CHAPITRE 4

CLASSIFICATION ET APPRENTISSAGE SUPERVISÉ

L'objectif de ce chapitre est de savoir si un algorithme d'apprentissage automatisé supervisé peut améliorer la détection des préoccupations transverses en utilisant des métriques de classe, dont le couplage et la cohésion. Les premières sections de ce chapitre présentent les jeux de données ainsi que les métriques à l'étude. On présente ensuite le principe de la classification. On présente ensuite la méthode du seuil de discrimination, permettant de classifier avec une seule métrique, puis on présente brièvement les algorithmes d'apprentissage automatisés utilisés. On présente ensuite les résultats, en incluant essentiellement les meilleurs résultats parmi les combinaisons testées. On compare finalement ces meilleurs résultats avec la méthode du seuil de discrimination pour voir si l'apprentissage automatisé améliore la détection des préoccupations transverses.

4.1 Jeux de données

Cette section présente les systèmes servant de jeux de données pour l'étude présentée dans ce mémoire.

4.1.1 JHotDraw

JHotDraw est un logiciel implémenté en Java, originellement en Smalltalk (HotDraw), qui a été largement utilisé dans l'étude des préoccupations transverses (Marin *et al.*, 2004; Ceccato *et al.*, 2006) et remanié en aspect en AspectJ pour fin de démonstration (Marin, 2004; Van Deursen, Marin & Moonen, 2005; Marin, Moonen & Deursen, 2007). La version aspectualisée est appelée AJHotDraw [1].

Pour le projet du présent mémoire, nous comparons le projet remanié, AJHotDraw, et la version 60b1 en Java de JHotDraw. Notre jeu de données est composé des classes de la version Java de

[1] https://sourceforge.net/projects/ajhotdraw/

JHotDraw. Les classes identifiées comme porteuses de préoccupations transverses sont celles qui ont été affectées par ce remaniement aspect. Plus précisément, il s'agit des classes dont le code a été migré vers un aspect ou des classes contenant des points de coupures (*joint point*) ciblés par au moins un aspect. Les préoccupations transverses reconnues dans JHotDraw sont entre autres *Undo* (défaire), la persistance et le patron observateur. La préoccupation transverse *undo* est répandue dans la hiérarchie des commandes. Chaque classe de commande hérite de `AbstractCommand` et contient une classe nichée statique `UndoActivity`. Cette classe nichée contient des méthodes pour défaire et refaire la commande.

4.1.2 HSQLDB

HSQLDB (Hyper Structured Query Language DataBase) est un logiciel de base de données relationnelle à code source ouvert [2] développé en Java. Il a été remanié en aspect [3] (Störzer, Eibauer & Schoeffmann, 2006). Le traçage (*logging*), la mise en cache et l'authentification sont des exemples de préoccupations transverses trouvées dans ce logiciel.

Dans notre recherche, le principe est le même que pour JHotDraw. On compare la version originelle et la version remaniée du logiciel et on marque les classes du logiciel originel qui ont été affectées par le remaniement dans la version remaniée.

4.1.3 Catalina

Catalina [4] est une portion du logiciel TOMCAT développé dans le projet Jakarta. Catalina est un conteneur de servlets. Catalina a été analysée pour en trouver des préoccupations transverses (Marin *et al.*, 2004). Plusieurs préoccupations transverses y ont été trouvées. Pour notre recherche, nous avons identifié principalement la journalisation et la gestion du cycle de vie, mais d'autres préoccupations ont été oubliées.

---

[2] https://hsqldb.org/

[3] https://sourceforge.net/projects/ajhsqldb/

[4] https://archive.apache.org/dist/tomcat/tomcat-5/archive/v5.0.24/

Le tableau 4.1 montre le nombre de classes ainsi que le nombre de classes identifiées comme porteuses d'au moins une préoccupation transverse pour les trois systèmes à l'étude. Remarquons que le système HSQLDB a plus de la moitié de ses classes porteuses d'une préoccupation transverse. L'un des aspects de HSQLDB concerne l'adoucissement d'exceptions. En Java, lorsqu'une méthode lance une exception, elle doit déclarer dans sa signature le type d'exception lancée avec le mot réservé `throws`. Cependant, les exceptions dérivées de la classe RuntimeException sont exemptées de cette exigence. Le remaniement aspect de HSQLDB utilise la fonctionnalité `declare soft` sur la classe d'exception `HsqlException` pour éviter de devoir envelopper chacune de ces exceptions dans une `RuntimeException`. Donc, chaque classe lançant une exception adoucie par un aspect est identifiée comme porteuse d'une préoccupation transverse.

Tableau 4.1 Pourcentage des classes ayant une préoccupation transverse selon les systèmes

Système	**Nombre de classes total**	**Nombre de classes avec préoccupation transverse**	**Pourcentage de classes ayant une préoccupation transverse**
JHotDraw	274	61	22.3%
HSQLDB	239	131	54.8%
Catalina	367	50	13.6%

4.2 Métriques de référence

Cette section présente les métriques utilisées pour les expérimentations et provenant de la littérature.

4.2.1 Fan-in

La métrique fan-in compte, pour chaque méthode, le nombre d'appels faits vers cette méthode dans le système. C'est le couplage entrant. Cette métrique est incluse dans l'outil FINT (Marin *et al.*, 2006). Cette métrique s'applique au niveau de granularité de la méthode. Dans le cadre du présent travail, qui se concentre sur le niveau de granularité de la classe, la métrique fan-in est évaluée pour une classe donnée en retenant le fan-in maximal de ses méthodes.

4.2.2 Fan-out

La métrique fan-out compte, pour chaque méthode, le nombre d'appels vers d'autres méthodes. C'est le couplage sortant. Elle est adaptée de la même manière que le Fan-in au niveau de la classe.

4.2.3 RandomWalk

Zhang et al. ont proposé la simulation de marches aléatoires pour trouver des préoccupations transverses (Zhang & Jacobsen, 2012). Il s'agit de construire un graphe conceptuel dirigé du système à partir des dépendances entre les classes. Ensuite, on simule des marches aléatoires dans le graphe à l'instar d'un programmeur qui explore le système en choisissant une classe au hasard et qui navigue en suivant les liens de couplage entrant et sortant. Chaque classe a un compteur d'utilisation et un compteur d'agrégation. À chaque transition, l'explorateur incrémente un compteur de la classe visitée selon le sens du lien de couplage suivi. Deux marches sont simulées. La marche d'utilisation suit le couplage sortant et la marche d'agrégation suit le couplage entrant.

Ces marches ne sont pas réalisées concrètement pour la simulation. La simulation est plutôt faite en calculant des probabilités en utilisant une chaine de Markov. Cela permet de calculer des métriques de façon déterministe. Le rapport entre l'utilisation d'une classe par rapport à son agrégation permet de faire ressortir les classes fortement sollicitées, mais qui n'en sollicitent que peu d'autres.

Pour le présent projet de recherche, une version simplifiée de cette approche a été implémentée. Pour l'instant, nous avons seulement implémenté la marche d'utilisation, faute de temps et en raison de la complexité de l'implémentation. On utilise les arbres syntaxiques construits à l'aide de l'outil Eclipse PDE pour construire le graphe conceptuel du système. Dans notre implémentation, une marche aléatoire d'utilisation est simulée pour chaque nœud du graphe. Pour chaque marche, tous les nœuds accessibles en suivant le couplage sortant sont visités

récursivement. À chaque visite, on ajoute au compteur la probabilité que cette classe soit visitée au cours de la marche.

4.3 Nouvelles métriques

Cette section présente les métriques que nous avons développées au cours de ce projet.

4.3.1 Remote-fan-in

Le Remote-fan-in est une version convoluée du fan-in. Les métriques `Remote-fan-in` `min`, `max` et `moyen` évaluent, pour une classe donnée, le fan-in de chaque classe dont une méthode est appelée. Les versions `min`, `max` et `moyen` retiennent respectivement le minimum, le maximum et la moyenne de ces valeurs. L'objectif de cette métrique est de mettre en évidence les classes qui appellent une classe fortement sollicitée. L'intuition pour cette métrique est corollaire à l'hypothèse qu'une classe fortement sollicitée implémente une préoccupation transverse. Cette préoccupation peut être cohésive dans la classe qui en implémente le cœur mais subordonnée dans les classes du cœur.

4.3.2 Cohésion d'interface

En supposant qu'une interface représente un rôle conceptuel, il en découle que si une classe est cohésive et implémente une interface, alors le rôle de l'interface est le rôle principal de la classe et ses méthodes servent à assumer ce rôle. Si une classe implémente une interface et qu'un faible pourcentage de ses méthodes l'implémentent, alors soit les autres méthodes sont des supports et la classe est cohésive, ou soit l'interface correspond à un rôle secondaire et donc la classe n'est pas cohésive. De plus, si une interface ne représente qu'un rôle secondaire dans toutes les classes qui l'implémentent, alors ce rôle est vraisemblablement une préoccupation transverse.

La cohésion des interfaces est une mesure que nous avons développée dans le cadre de cette recherche. Elle évalue, pour une classe donnée, le pourcentage des méthodes de la classe qui implémente une même interface. Dans le cas où une interface est implémentée par la classe,

la cohésion est calculée séparément pour chaque interface et la cohésion minimale est retenue. Si la classe n'implémente aucune interface, la métrique prend pour valeur -1. Le rôle de cette métrique est de mettre en évidence les classes qui implémentent une interface qui représente un rôle secondaire. Le principe de la cohésion est qu'une classe n'aie qu'un seul rôle principal et que ses méthodes servent à assumer ce rôle. On soupçonne donc les interfaces occupant une faible portion des classes de représenter une préoccupation transverse. Comme elle est formulée, la métrique ne met pas bien les classes peu cohésives en évidence, parce que les classes sans interface ont la valeur -1, qui est inférieure aux autres valeurs à mettre en évidence.

Pour contourner ce problème, on applique une fonction pour séparer la cohésion d'interface en catégories (formule ci-dessous). Il s'agit d'envoyer les valeurs -1 vers la catégorie supérieure pour que les catégories inférieures regroupent les classes avec une faible cohésion d'interface.

$$\text{Catégorie}(x) = \begin{cases} 1 & \text{si} \quad 0 \leq x < 25\% \\ 2 & \text{si} \quad 25\% \leq x < 50\% \\ 3 & \text{si} \quad 50\% \leq x < 75\% \\ 4 & \text{si} \quad x \geq 75\% \\ 5 & \text{si} \quad x = -1 \end{cases} \tag{4.1}$$

4.3.3 Profondeur de hiérarchie

La profondeur de hiérarchie mesure la profondeur de l'arbre des super types d'une classe. Cela inclue les classes mères, grand-mères (etc.) ainsi que les interfaces et les super interfaces. La classe `Object` est exclue.

4.3.4 Cohésion de la classe

On utilise les métriques de cohésion présentées au chapitre 3. Les métriques COHCL et nbCluster sont les métriques proposées par (Sadaoui *et al.*, 2012). COHCL2 et nbCluster2 sont issues d'un

projet de recherche poursuivant l'exploration de l'utilisation de la classification hiérarchique ascendante pour mesurer la cohésion.

4.4 Classification

Une classification est une séparation du jeu de données en catégories. Dans le contexte de ce travail, les catégories sont celle des positifs et celle des négatifs quant à la présence de préoccupation transverse dans la classe logicielle. Les éléments positifs sont les classes logicielles dans lesquelles une préoccupation transverse a été identifiée. On peut évaluer la qualité d'une classification en la comparant à une classification de référence. Le tableau 4.2 présente un modèle de matrice de confusion.

Tableau 4.2 Modèle de matrice de confusions

	Positifs (prédiction)	**Négatifs Prédiction**	**Somme**
Positifs (référence)	VP	FN	VP + FN
Négatifs (référence)	FP	VN	FP + VN
Somme	VP + FP	FN + VN	Total

Le tableau 4.3 présente les éléments de la matrice de confusion et quelques indices, calculés à partir de ceux-ci, permettant d'évaluer la qualité d'une classification. La précision représente le pourcentage des éléments prédits positifs qui sont positifs dans la classification de référence. La précision peut servir à répondre à la QR2. Le rappel représente le pourcentage des éléments positifs, dans la classification de référence, qui sont prédits positifs. Le rappel peut servir à répondre à la QR1 s'il y a une différence significative. Cependant, si deux classifications ont le même rappel, cet indicateur n'informe pas sur la complémentarité des éléments détectés. Le taux d'erreur représente la proportion des éléments mal identifiés. Le G-Mean est une moyenne géométrique entre la précision et le rappel. Cet indice est aussi connu sous le nom d'indice de Fowlkes-Mallows (Halkidi, Batistakis & Vazirgiannis, 2001).

Le carré du G-Mean borne le rappel et la précision. En effet, le rappel et la précision sont compris dans l'intervalle $[0, 1]$. Le cas où le G-Mean est de 0 est trivial et dégénéré. Si le G-Mean n'est pas nul, alors la précision et le rappel sont strictement positifs. On a donc

$$\begin{aligned}
\sqrt{\text{rappel}} &= \frac{\text{GMean}}{\sqrt{\text{précision}}} \\
\Rightarrow \text{rappel} &= \frac{\text{GMean}^2}{\text{précision}} \geq \frac{\text{GMean}^2}{1} = \text{GMean}^2 \\
\Rightarrow \text{rappel} &\geq \text{GMean}^2.
\end{aligned}$$

On peut montrer d'une façon similaire que la précision est aussi bornée par le carré du G-Mean. Pour autant que le G-Mean soit suffisamment élevé, il peut servir à répondre à la QR1 et à la QR2.

Tableau 4.3 Indicateurs de la qualité d'une classification

Élément	Formule	Description
VP	-	Nombre de vrais positifs
VN	-	Nombre de vrais négatifs
FP	-	Nombre de faux positifs
FN	-	Nombre de faux négatifs
Précision	$\text{Précision} = \frac{\text{VP}}{\text{VP} + \text{FP}}$	Proportions des éléments classés positifs qui le sont
Rappel	$\text{Rappel} = \frac{\text{VP}}{\text{VP} + \text{FN}}$	Proportion des éléments positifs étant classés comme positifs
Taux d'erreur	$\text{Taux d'erreur} = \frac{\text{FP} + \text{FN}}{\text{VP} + \text{VN} + \text{FP} + \text{FN}}$	Proportions des éléments mal identifiés
G-Mean	$\text{GMean} = \sqrt{\text{Rappel} \cdot \text{Précision}}$	Indice de Fowlkes–Mallows

Le taux d'erreur est moins intéressant, car lorsque les éléments positifs sont minoritaires, il peut donner une valeur élevée même avec un taux de rappel faible. Dans ce chapitre, on utilise principalement le G-Mean.

Le reste de ce chapitre présente la recherche de classificateurs maximisant le G-Mean. On s'intéresse particulièrement aux métriques et aux combinaisons de métriques utilisées pour ces classificateurs. La prochaine section traite d'une méthode utilisant des seuils de discrimination avec une seule métrique à la fois. La section 4.6 traite d'apprentissage automatisé combinant plusieurs métriques.

4.5 Seuil de discrimination

La classification par seuillage consiste à sélectionner une métrique m et un seuil σ et à retenir comme positifs les éléments x_i tels que $m(x_i) \geq \sigma$ (ou $m(x_i) \leq \sigma$, selon le cas).

Pour une métrique donnée, on peut trouver un seuil qui maximise le G-Mean en faisant une classification pour chaque valeur de la métrique présente dans un jeu de données. On ne s'attend pas à pouvoir extrapoler le seuil optimal d'une métrique à un autre jeu de données. Il s'agit plutôt de considérer le G-Mean maximal. Une valeur élevée indiquerait un lien fort entre la métrique et la présence de préoccupation transverse. Une valeur faible ne permet pas de conclure une absence de lien.

La figure 4.1 montre l'évolution de la précision, du rappel et du G-Mean en fonction du seuil avec le fan-in pour le système JHotDraw. Les classes logicielles sont prédites positives lorsque leur valeur de fan-in est supérieure ou égale au seuil. Le G-Mean atteint une valeur maximale de 56.9% lorsque la valeur du seuil est 7. Ce seuil correspond aussi à la valeur maximale de la précision, soit de 49.4%. Le seuil minimal 0 inclue toutes les classes logicielles comme étant positives dans la prédiction. On peut donc déduire que la précision à ce seuil, soit de 22.3%, correspond à la proportion de classes logicielles positives dans la classification de référence.

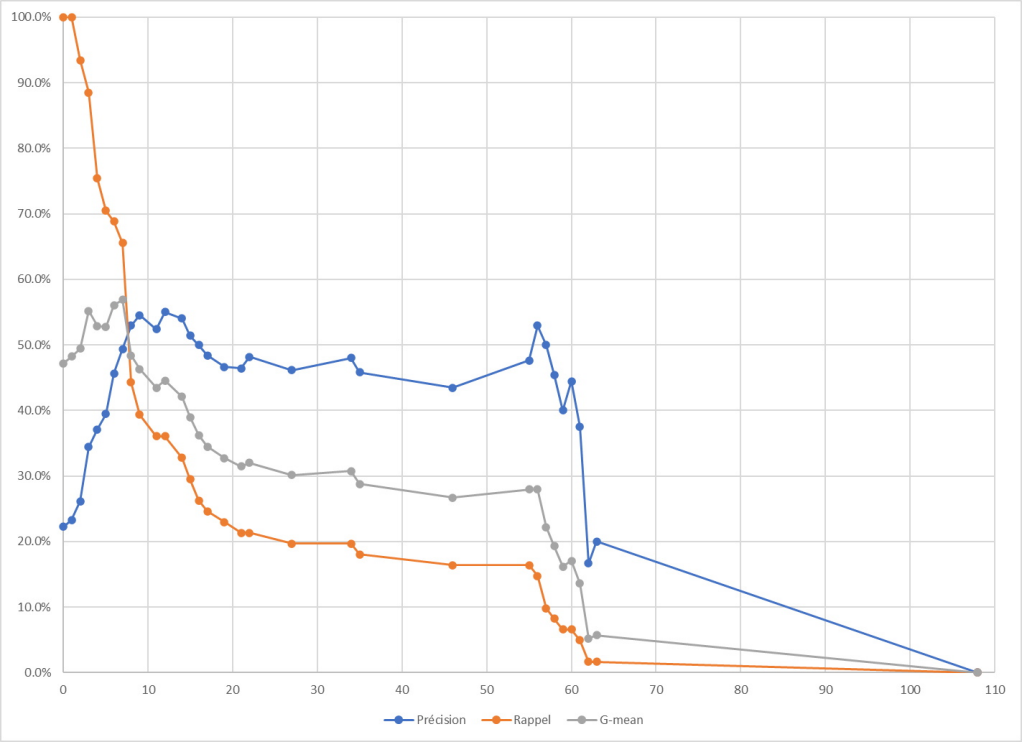

Figure 4.1 Seuillage du fan-in (JHotDraw)

La figure 4.2 montre un graphique similaire pour le système HSQLDB. La proportion des classes positives dans la classification de référence est de 54.0%. La valeur maximale du G-Mean est de 73.5% au seuil de 0. Il n'est donc pas avantageux dans ce cas d'utiliser un seuil comme modèle prédictif. Toutefois, le pic de précision avec les seuils plus élevés indique que les classes logicielles avec les valeurs de fan-in les plus élevées sont investies de préoccupations transverses.

La figure 4.3 montre un G-Mean dépassant 90% pour le système Catalina. Toutefois, comme l'identification des classes a été faite manuellement par moi-même et que cette identification est probablement incomplète et parfois arbitraire, il faut garder une réserve sur la validité de ce résultat et donc le considérer à titre indicatif seulement. Si on présume que l'identification des classes positives est fiable mais incomplète, il en découle que le rappel peut être surestimé mais que la précision peut être sous-estimée. Comme la précision augmente très rapidement selon la valeur du fan-in, on peut quand même confirmer que le fan-in est un indicateur précis. La précision plus élevée du fan-in pour ce système indique qu'il y a moins de faux positifs parmi

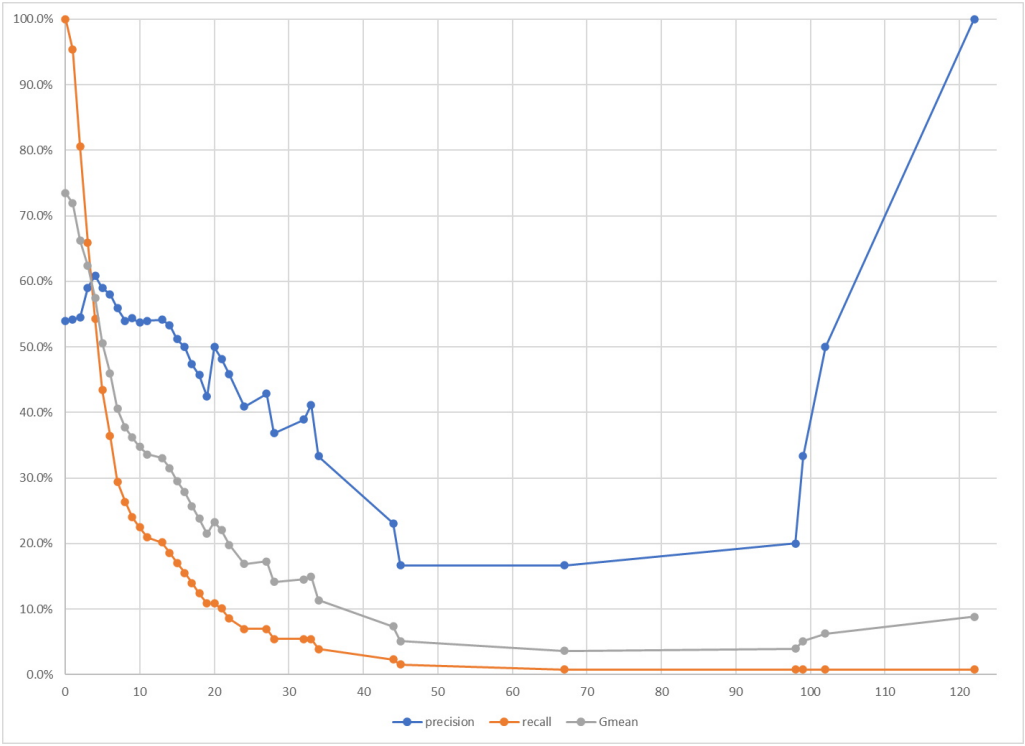

Figure 4.2 Seuillage du fan-in (HSQLDB)

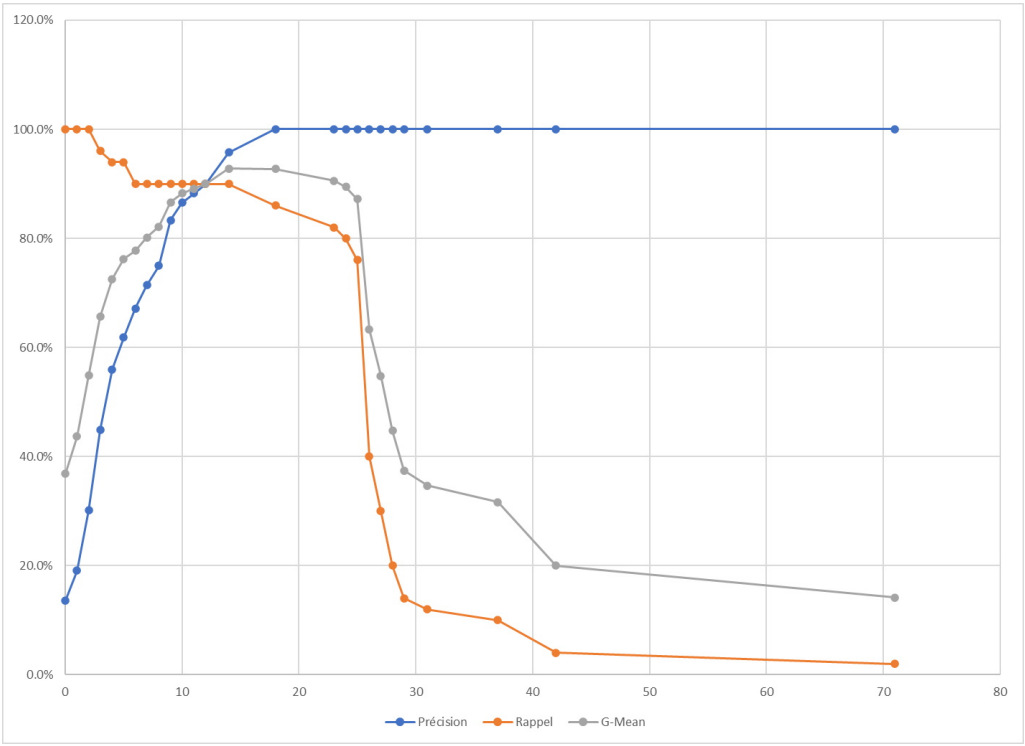

Figure 4.3 Seuillage du fan-in (Catalina)

les classes ayant un fan-in plus élevé. Parmi les faux positifs dans JHotDraw, il y a des méthodes qui sont appelées plusieurs fois, mais par des méthodes de la même classe. Dans ce cas, le fan-in est élevé alors que la classe n'est pas couplée à beaucoup d'autres classes. Il ne s'agit donc pas forcément de préoccupations transverses.

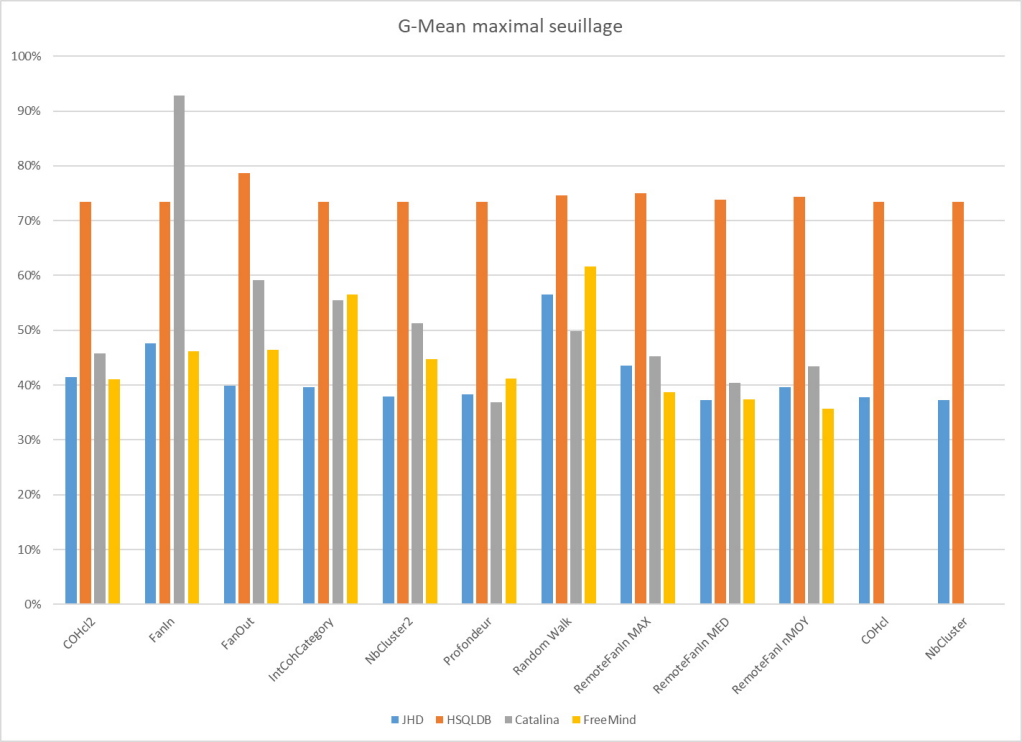

Figure 4.4 Maxima des G-Means de l'analyse par seuillage pour chaque système et chaque métrique

La figure 4.4 montre les valeurs maximales du G-Mean pour chaque système et chaque métrique. Pour JHotDraw, le RandomWalk est la seule métrique à dépasser 50% de G-Mean. Pour HSQLDB, les résultats varient très peu d'une métrique à l'autre et c'est le Fan-Out qui produit le meilleur G-Mean. Pour Catalina, le Fan-in dépasse 90% de G-Mean alors que les autres métriques produisent un G-Mean inférieur à 60%. Avec le système FreeMind, le RandomWalk se démarque encore et est suivi de la cohésion des interfaces.

Ces résultats montrent le caractère disparate des systèmes étudiées quant à la relation entre les classes identifiées positives et les différentes métriques. Cela suggère que même si on arrivait à

entrainer un classificateur sur un système avec de meilleures performances, ce classificateur risque d'être moins performant (voir inutile selon le degré de disparité) sur un autre système.

4.6 Apprentissage automatisé

L'apprentissage automatisé a été largement utilisé dans la dernière décennie dans de multiples domaines d'application. L'apprentissage automatisé est exploré dans la présente recherche pour tenter de répondre à la question suivante : Existe-t-il un algorithme d'apprentissage automatisé capable de détecter, sur la base de métriques de classes logicielles, la présence de préoccupations transverses ? Cette question servira à répondre à la QR3

Le logiciel Tanagra est utilisé dans le cadre de la présente recherche pour expérimenter l'utilisation d'algorithmes d'apprentissage automatisé supervisé. Tanagra est un logiciel de forage de données, gratuit et à utilisation libre. Il offre une interface graphique et quelques algorithmes d'apprentissage automatique [5].

Un jeu de données est défini par un ensemble de variables descriptives et un ensemble d'entrées auxquelles sont associées une valeur pour chacune des variables. Dans la présente étude, les entrées représentent chacune des classes du système et les variables comprennent le nom de la classe ainsi que les différentes métriques étudiées.

Un algorithme d'apprentissage supervisé peut être entrainé sur un jeu de données pour prédire une variable cible à partir d'autres variables. En l'occurrence, on entraîne ces algorithmes à prédire la présence ou non de préoccupations transverses à partir d'une combinaison de métriques. Le résultat de l'apprentissage est un classificateur capable d'offrir une prédiction sur n'importe quel élément à partir des valeurs de ses métriques.

Les expérimentations de cette recherche utilisent la technique de validation croisée. La validation croisée consiste à séparer le jeu de données pour entrainer l'algorithme sur une partie et tester les performances sur la partie restante. Cela sert à confirmer que l'algorithme ne fait pas de

[5] http://eric.univ-lyon2.fr/~ricco/tanagra/fr/tanagra.html

sur-apprentissage. En effet, un algorithme pourrait très bien être capable de bien classifier tous les éléments du jeu d'entraînement, sans être capable de bien classifier de nouvelles données.

La validation croisée à 10 plis (Kohavi, 1995), utilisée dans ce travail, consiste à séparer aléatoirement le jeu de données en 10 sections de taille égale. Itérativement, pour chacune de ces sections, on l'isole et on fait un apprentissage sur le reste du jeu de données. On effectue ensuite une prédiction sur la section isolée de l'itération actuelle. Ce procédé est appliqué pour chacune des 10 sections et on retient la somme des 10 matrices de confusion issues des prédictions.

Les paragraphes qui suivent présentent sommairement les différents algorithmes disponibles dans Tanagra et qui ont été utilisés pour cette recherche. À moins d'indications contraires, ces algorithmes ont été utilisés avec leurs configurations par défaut dans Tanagra. Il n'est pas envisageable de tester toutes les combinaisons par la force brute. Le nombre de combinaisons de métriques croît déjà de façon exponentielle selon le nombre de métriques disponibles. À cela se multiplient les différents paramètres de certains algorithmes. Les expérimentations sont davantage centrées sur les différentes combinaisons des variables descriptives (métriques) que sur les réglages particuliers des algorithmes.

4.6.1 Perceptron multicouche

Le perceptron est un algorithme d'apprentissage supervisé inventé par Rosenblatt en 1958 (Mehrotra, Mohan & Ranka, 1997) représentant un réseau de neurones artificiels. Le neurone artificiel, inspiré du fonctionnement du neurone biologique, est une fonction d'activation munie d'un biais qui produit une valeur (stimulus de sortie) en fonction des paramètres (stimuli d'entrée) et du biais. L'apprentissage consiste à modifier itérativement les biais pour minimiser une certaine fonction d'erreur. Le perceptron est originellement composé d'une seule couche de neurones et est limité à pouvoir classifier efficacement des données linéairement séparables. La rétropropagation proposée par Webos en 1974 permet l'utilisation de couches intermédiaires. Cela permet de dépasser la limitation du perceptron aux problèmes linéairement séparables (Mehrotra *et al.*, 1997).

L'outil Tanagra offre un algorithme d'un perceptron ayant une couche cachée optionnelle [6]. Le nombre de neurones dans cette couche peut être paramétré et est réglé à 10 par défaut.

4.6.2 Forêt d'arbres aléatoires

La forêt d'arbres aléatoires consiste à utiliser la méthode du *bagging* avec un ensemble d'arbres aléatoires (Breiman & Cutler, date inconnue). Le *bagging* combine un ensemble de modèles d'apprentissages (arbres aléatoires en l'occurrence) entraînés séparément et fournit une prédiction en faisant voter chacun des modèles prédictifs. Les arbres aléatoires sont des arbres décisionnels construits à partir d'un échantillon aléatoire du jeu de données ainsi qu'un sous-ensemble aléatoire de variables descriptives.

4.6.3 Classificateur bayésien naïf

L'algorithme du classificateur bayésien naïf utilise le théorème de bayes et repose sur l'hypothèse que les variables descriptives sont indépendantes entre elles. Toutefois, ce classificateur peut donner de bons résultats même si les variables descriptives ne sont pas totalement indépendantes (Domingos & Pazzani, 1997).

4.6.4 K plus proches voisins (K-NN)

La méthode des K plus proches voisins, ou K-NN (K nearest neighbours), consiste à prendre les K plus proches voisins d'une valeur à classer. Ces K voisins votent pour leur classe respective et la classe ayant le plus de votes est retenue comme prédiction. Dans Tanagra, la valeur de K est de 5 par défaut. Dans Tanagra, la métrique de distance est HVDM (Heterogenous Value Difference Metric) (Wilson & Martinez, 1997).

---

[6] https://data-mining-tutorials.blogspot.com/2017/12/configuration-of-multilayer-perceptron.html?m=1

4.6.5 Support Vector Machine

Le SVM (Support Vector Machine) consiste à projeter les données dans un espace de dimension supérieure et à trouver un hyperplan qui sépare les échantillons de façon à maximiser la marge entre ces échantillons (Schölkopf, Smola & Bach, 2002). La classification d'une nouvelle donnée consiste à la projeter dans cet espace et à attribuer la classe selon le côté de l'hyperplan où elle se situe.

4.6.6 Régression Logistique

La régression logistique est un modèle de régression pouvant être appliqué lorsque la variable à prédire est binaire (Hosmer, Lemeshow & Sturdivant, 2013).

4.7 Sommaire des résultats et conclusion

Les expérimentations sont orientées sur les combinaisons de métriques. Pour chaque combinaison de métriques, un classificateur est entrainé avec chacun des algorithmes présentés plus haut. Le tableau 4.4 montre un sommaire des expérimentations les plus prometteuses avec JHotDraw et HSQLBD. Ces combinaisons ont aussi été testées avec Catalina à titre indicatif. Pour fin de concision, on présente seulement l'algorithme ayant maximisé le G-Mean. Avec JHotDraw, le plus haut G-Mean a été obtenu en utilisant le 5-NN avec le fan-in et notre implémentation partielle du RandomWalk.

Avec HSQLDB, le plus haut G-Mean est obtenu avec le fan-out, NbClusters et l'algorithme du perceptron. Pour ce système, les classifications sont moins performantes lorsque le fan-out n'est pas inclus dans la combinaison. Pour ces combinaisons de métriques, il semble que le perceptron performe mieux avec HSQLDB qu'avec JHotDraw. Les résultats avec Catalina sont systématiquement supérieurs à 93%.

Le tableaux 4.5 compare les classifications optimales de la méthode de seuillage et des apprentissages automatisés. Pour nos deux principaux jeux de données, l'apprentissage automatisé

Tableau 4.4 G-Mean maximal par combinaison de métrique pour chaque système

Métriques	Algorithme	G-Mean maximal		
		JHotDraw	**HSQLDB**	**Catalina**
Fan-in, NbClusters	Forêt aléatoire	45.2%	60.8%	98.2%
Fan-in, cohésion d'interface	5-NN	43.0%	58.6%	98.9%
Fan-in, profondeur	5-NN	45.8%	67.3%	98.4%
Fan-in, RemoteFanInMax	Random Forest	49.2%	69.8%	99.0%
Fan-in, RandomWalk	5-NN	**54.6%**	68.5%	98.1%
Fan-in, Fan-out	Perceptron	24.8%	77.3%	98.7%
Fan-in, Fan-out	Forêt aléatoire	45.4%	70.8%	98.1%
Fan-in, Fan-out, NbClusters	5-NN	42.3%	75.0%	**99.0%**
Fan-in, Fan-out, NbClusters	Perceptron	26.8%	77.5%	98.6%
Fan-out, NbClusters	Perceptron	24.2%	**79.2%**	93.3%
Fan-out, cohésion d'interface	5-NN	42.2%	77.5%	94.0%

n'améliore pas significativement le G-Mean. Cela permet de répondre positivement à la QR2 : on peut augmenter la précision en combinant des métriques. La QR1 est validée seulement avec le système Catalina, car le rappel et la précision sont augmentés. La baisse du rappel avec JHotDraw et HSQLBD ne peut pas confirmer ni infirmer la QR1. Même avec un rappel plus faible, on ne peut pas en déduire si les éléments détectés sont complémentaires. Il serait intéressant de chercher les classes les moins bien classées pour questionner la présence d'une préoccupation transverse ou chercher un indice trahissant mieux la présence ou l'absence d'une préoccupation transverse. Malheureusement, Tanagra ne permet pas facilement d'identifier les éléments selon leur position dans la matrice de confusion.

Pour la QR3, à savoir si l'apprentissage supervisé peut améliorer la détection des préoccupations transverses, la réponse n'est pas aussi simple. D'une part, l'apprentissage automatisé a permis de confirmer la contribution de l'apprentissage sur la précision. Cependant, pour que l'apprentissage supervisé puisse avoir une utilité concrète, il faut qu'un apprentissage supervisé sur un système permette de faire une prédiction sur un autre système. La disparité des contributions des métriques sur les différents systèmes laisse présager que la généralisation d'un apprentissage

d'un système à un autre n'est pas possible (difficile à la limite). L'utilité de l'apprentissage est donc limitée à chercher des corrélations entre les variables descriptives et la variable à prédire.

Pour la QR4, l'utilisation de la cohésion améliore la précision avec le système HSQLBD, mais les résultats avec JHotDraw ne sont pas aussi concluants. Bien que nos questions de recherche utilisent le couplage entrant comme méthode de référence, le couplage sortant semble plus utile que le couplage entrant avec HSQLBD.

Tableau 4.5 Comparaison du seuillage et de l'apprentissage automatisé

Système	**Approche**	**Métriques**	**Algorithme**	**G-Mean**	**Précision**	**Rappel**
JHotDraw	Seuillage	Fan-in	-	56.9%	49.4%	65.6%
	Apprentissage	Fan-in, RandomWalk	5-NN	54.6%	61.7%	48.3%
HSQLBD	Seuillage	Fan-out	-	78.8%	64.6%	96.2%
	Apprentissage	Fan-out, NbClusters	Perceptron	79.2%	80.5%	78.0%
Catalina	Seuillage	Fan-in	-	92.8%	95.7%	90.0%
	Apprentissage	Fan-in, Fan-out, NbClusters	5-NN	99.0%	98.1%	100.0%

Considérant que l'apprentissage supervisé ne semble pas exploitable pour créer un modèle prédictif pouvant être généralisé, nous abandonnons cette piste. La suite de cette recherche, présentée dans le prochain chapitre, étudie entre autres la complémentarité des métriques quant à la détection des préoccupations transverses afin de mieux répondre à la QR1.

CHAPITRE 5

DIVISER POUR MIEUX RÉGNER

Après les expérimentations sur l'apprentissage automatisé, la question de comprendre la complémentarité de la contribution des métriques, entre autres la cohésion, reste ouverte. Le présent chapitre a pour but de présenter une piste heuristique pour comprendre la contribution des métriques et mettre en évidence les éléments qui ne sont pas bien détectés.

Pour chaque jeu de donnés à l'étude, l'identification des préoccupations transverses est possiblement incomplète. Au risque de redondance, rappelons qu'il faut donc toujours tenir compte que le rappel peut être surestimé. Une itération supplémentaire de recherche de préoccupations transverses a donc le potentiel de modifier les résultats. Par soucis de consistance, dans le cadre de ce mémoire, toutes les expérimentations présentées pour un jeu de données sont basées sur la même identification des préoccupations transverses.

5.1 Outil d'analyse

Avant d'aborder directement les expérimentations, nous nous sommes intéressés aux limitations techniques rencontrées précédemment. Un des goulots d'étranglement les plus importants était l'utilisation de l'outil Tanagra. Chaque expérimentation nécessitait de configurer manuellement la sélection des variables et de copier les résultats dans un classeur. Cela nous a grandement limité dans la quantité de combinaisons testées. À un certain moment, nous avons discuté et ajusté l'identification des classes porteuses de préoccupations transverses pour tenir compte des classes affectées par un aspect. Il a fallu recommencer les expérimentations. Cette rigidité est problématique pour l'exploration d'autres systèmes comme Catalina où l'identification des préoccupations transverses est progressive.

L'automatisation des expérimentations a donc été une priorité pour la suite de ce projet. D'une part, nous avons adapté notre système de données afin de pouvoir modifier plus facilement l'identification des préoccupations transverses. Notre système permet d'identifier des classes porteuses en leur ajoutant des mots-clés. Ces mots-clés peuvent par la suite être inclus ou exclus

de l'identification au besoin. Le système construit la variable cible à prédire selon les mots-clés inclus. Cela permettra ultérieurement d'analyser les préoccupations transverses spécifiquement et globalement. D'autres part, les expérimentations abordées dans la suite de ce chapitre ont été programmées directement dans l'outil de sorte à être complètement automatisées.

5.2 Précision maximale

L'heuristique de ce chapitre consiste à analyser, pour chaque classe investie de préoccupation transverse, la précision maximale avec laquelle elle est détectée par l'une des classifications. Il s'agit alors d'étudier les classes qui sont détectées avec moins de précision et possiblement trouver de nouvelles pistes.

Contrairement au rappel, la précision n'est pas surestimée, à condition que les classes identifiées comme porteuses de préoccupations transverses le soient vraiment. Une façon simple de combiner deux classifications est d'en faire l'union. C'est-à-dire qu'un élément est prédit positif si au moins l'une des deux classifications le prédit positif. On peut montrer que si la précision de chaque classification est bornée par α, alors la précision de la classification d'union est bornée par $\frac{\alpha}{2-\alpha}$.

Démonstration

Soit A et B deux classifications telles que $Prec_A, Prec_B > 0$, où $Prec_A$ désigne la précision de la classification A et $Prec_B$ désigne la précision de la classification B. Notons $A \cup B$ la classification d'union. Considérons une borne α arbitraire telle que $0 < \alpha \leq \min(Prec_A, Prec_B)$. Donc,

$$\begin{aligned} Prec_A \geq \alpha &\Rightarrow \frac{VP_A}{VP_A + FP_A} \geq \alpha \\ &\Rightarrow VP_A \geq \alpha \left(VP_A + FP_A\right) \\ &\Rightarrow (1-\alpha)\, VP_A \geq \alpha \cdot FP_A \\ &\Rightarrow FP_A \leq \left(\frac{1}{\alpha} - 1\right) VP_A. \end{aligned}$$

De façon similaire, on a aussi $FP_B \leq \left(\frac{1}{\alpha} - 1\right) VP_B$. Ensuite, les classifications A et B peuvent partager ou non des éléments communs. Il faut donc tenir compte des inégalités suivantes :

$$\begin{aligned} 0 \leq VP_A \leq \max\left(VP_A, VP_B\right) \leq VP_{A\cup B} \leq VP_A + VP_B \\ FP_{A\cup B} \leq FP_A + FP_B \end{aligned}$$

Sans perte de généralité, on suppose que $VP_A \geq VP_B$. On trouve donc

$$\begin{aligned} VP_{A\cup B} \leq VP_A + VP_B \leq 2VP_A \\ VP_A \leq VP_{A\cup B} \\ VP_B \leq VP_{A\cup B}. \end{aligned}$$

La précision de la classification unie est $Prec_{A\cup B} = \frac{VP_{A\cup B}}{VP_{A\cup B} + FP_{A\cup B}}$. En utilisant les inégalités précédentes, on peut déduire une borne inférieure à la précision :

$$
\begin{aligned}
Prec_{A\cup B} &= \frac{VP_{A\cup B}}{VP_{A\cup B} + FP_{A\cup B}} \geq \frac{VP_A}{VP_{A\cup B} + FP_{A\cup B}} \\
&\geq \frac{VP_A}{VP_A + FP_{A\cup B}} \geq \frac{VP_A}{VP_A + FP_A + FP_B} \\
&\geq \frac{VP_A}{VP_A + \left(\left(\frac{1}{\alpha} - 1\right) VP_A\right) + \left(\left(\frac{1}{\alpha} - 1\right) VP_B\right)} \\
&= \frac{VP_A}{VP_A + \left(\frac{1}{\alpha} - 1\right)(VP_A + VP_B)} \geq \frac{VP_A}{VP_A + \left(\frac{1}{\alpha} - 1\right)(2VP_A)} \\
&= \frac{VP_A}{VP_A + \left(\frac{2}{\alpha} - 2\right)(VP_A)} = \frac{VP_A}{VP_A\left(1 + \frac{2}{\alpha} - 2\right)} \\
&= \frac{1}{1 + \frac{2}{\alpha} - 2} = \frac{1}{\frac{2}{\alpha} - 1} = \frac{\alpha}{2 - \alpha} \\
&\Rightarrow Prec_{A\cup B} \geq \frac{\alpha}{2 - \alpha} \quad \square
\end{aligned}
$$

En suivant un raisonnement similaire, on peut généraliser le principe et faire l'union de N classifications dont la précision est supérieure à α. On trouve alors que la précision est bornée par $\frac{\alpha}{N - \alpha(N-1)}$.

Il s'agit du pire des cas, où tous les classificateurs détectent seuls les mêmes vrais positifs et des faux positifs différents. Il est donc pertinent d'unir des classifications précises, pourvu que ces classifications ne détectent pas seulement les mêmes classes. On peut alors chercher des classificateurs précis qui, ensemble, détectent conjointement un maximum de vrais positifs.

Les travaux présentés dans ce chapitre se limitent à utiliser les classificateurs par seuil de discrimination. Pour une classe identifiée comme porteuse de préoccupation transverse, il s'agit

de faire une classification avec chaque seuil, de retenir les classifications qui détectent la classe comme positive, et de retenir la précision maximale de ces classifications. Cela permet de voir, pour chaque classe, quelle métrique la détecte avec le plus de précision. Cela permettra de répondre à la QR1. Si une métrique permet de détecter une classe avec une précision supérieure à d'autres métriques, alors il y a une complémentarité.

Cette analyse a été programmée dans notre outil, qui produit un tableau listant les précisions maximales de chaque paire classe-métrique.

5.3 Résultats

Cette section présente les résultats de l'analyse des précisions maximales avec JHotDraw et HSQLDB. Cette analyse se penche dans un premier temps sur les classes détectées avec le plus de précision afin de comparer la complémentarité des métriques. Dans un deuxième temps, les classes détectées avec le moins de précision sont présentées.

5.3.1 JHotDraw

La figure 5.1 présente les précisions maximales obtenues par le fan-in, le fan-out, la cohésion (nbCluster) et la marche aléatoire. Seules les classes dont la précision maximale est d'au moins 75% sont représentées. Le couplage sortant est plus précis que le couplage entrant. Le couplage entrant a une précision maximale de seulement 55%. La marche aléatoire, le nombre de clusters et le couplage sortant ont chacune des précisions maximales exclusives sur quelques classes. Cela indique qu'elles ont bien une pertinence complémentaire, ce qui répond positivement à la QR1. La précision avec NbCluster est presque toujours plus précise avec la première version de la métrique.

La figure 5.2 présente les précisions maximales obtenues en incluant la cohésion des interfaces (définie à la section 4.3.2), qui donne des résultats intéressants. La première version de la mesure donne une précision inférieure à 30%. Mais en utilisant la version avec les catégories, la précision grimpe à 92% pour plusieurs classes. On essaie finalement de filtrer le fan-in avec la

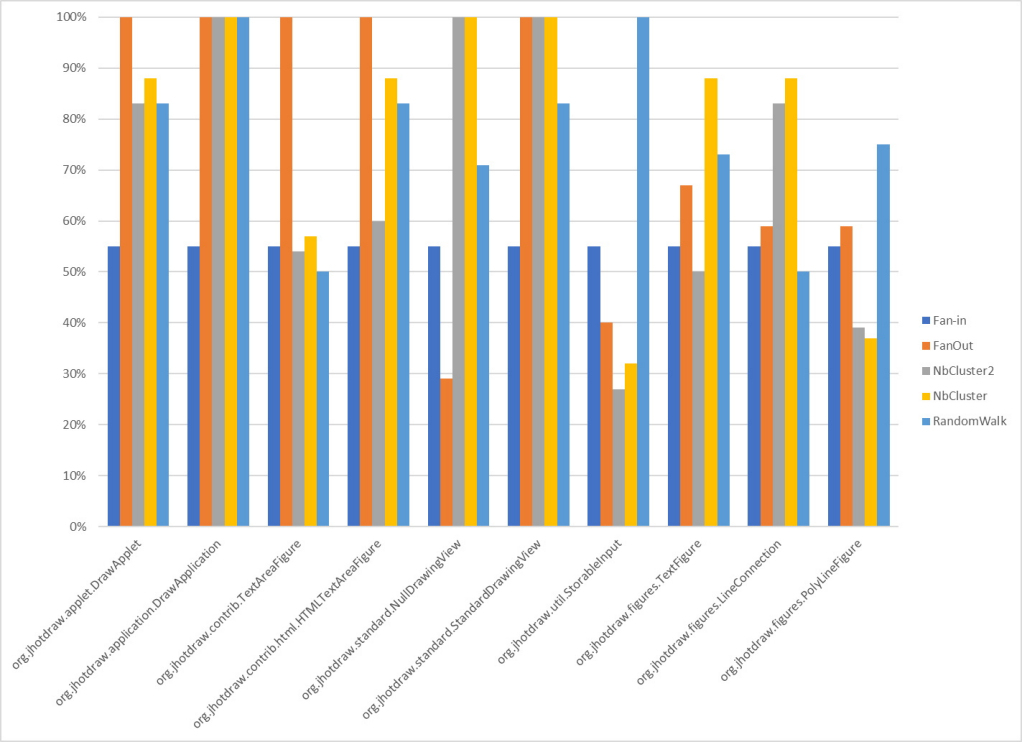

Figure 5.1 Précisions maximales classes détectées avec précision ≥ 75%, JHotDraw

catégorie de cohésion d'interfaces. La précision atteint alors 100% pour 10 classes, dont cinq avec une exclusivité par rapport aux autres métriques.

$$\text{Filtre}_{\text{FanIn,Cohésion}}(c) = \begin{cases} \text{FanIn}(c) & si\ \text{Catégorie}\ (\text{interfaceCohésion}(c)) = 1 \\ 0 & sinon \end{cases} \tag{5.1}$$

Ces résultats confirment que le couplage entrant, le couplage sortant, la cohésion et la marche aléatoire peuvent apporter une contribution complémentaire à la détection des préoccupations transverses avec le système JHotDraw.

Cette analyse permet du même souffle d'identifier les classes les plus difficiles à détecter. Le tableau 5.1 présente les dix classes détectées avec le moins de précision. Parmi les classes les plus difficiles à détecter, il y a les classes `UndoActivity`. Ce sont des sous-classes de la plupart

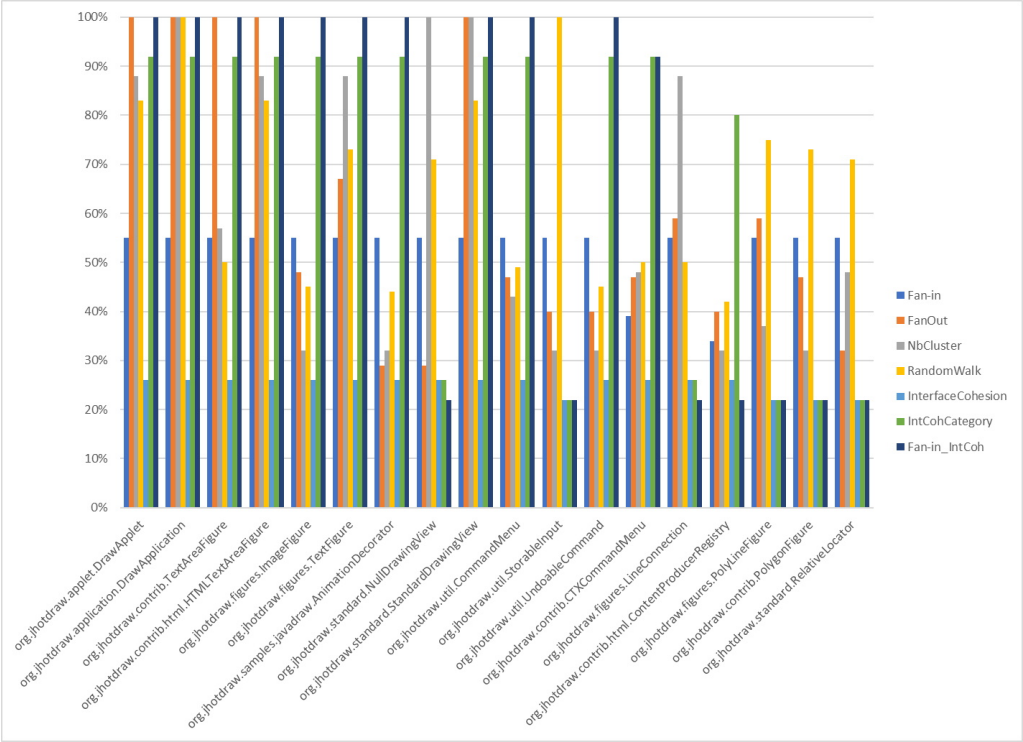

Figure 5.2 Précisions maximales avec la cohésion d'interfaces avec JHotDraw

des classes de la hiérarchie d'`AbstractCommand`. Elles servent à définir comment chaque commande peut être défaite, et refaite. Bien que ces classes `UndoActivity` représentent bien une préoccupation transverse, chaque implémentation est spécifique à la commande à défaire. On ne peut donc pas la factoriser directement en aspect.

La classe `ArrowTip` est affectée par le remaniement aspect uniquement par l'adoucissement des méthodes de lecture qu'elle utilise (voir figure 5.3) et c'est pour cela qu'elle était identifiée positive dans le jeu de données. Toutefois, les méthodes `read(StorableInput)` et `write(StorableOutput)` représentent la préoccupation transverse de persistance. La mesure de cohésion des interfaces a échoué à la détecter parce que l'implémentation de l'interface est indirecte. L'interface Storable est implémentée par la classe mère d'`ArrowTip`, `AbstractLineDecoration`, qui implémente l'interface `LineDecoration` qui elle hérite de `Storable`. La mesure de cohésion d'interface devrait donc être reprogrammée pour tenir compte

Tableau 5.1 Dix classes de JHotDraw détectées avec moins de précision

Classe	Précision maximale
org.jhotdraw.contrib.html.TextHolderContentProducer	42%
org.jhotdraw.standard.AbstractCommand.EventDispatcher	42%
org.jhotdraw.figures.ArrowTip	40%
org.jhotdraw.contrib.html.ResourceContentProducer	40%
org.jhotdraw.standard.AlignCommand.UndoActivity	40%
org.jhotdraw.contrib.html.URLContentProducer	38%
org.jhotdraw.figures.ConnectedTextTool.DeleteUndoActivity	38%
org.jhotdraw.standard.DeleteCommand.UndoActivity	38%
org.jhotdraw.standard.BringToFrontCommand.UndoActivity	35%
org.jhotdraw.standard.ChangeAttributeCommand.UndoActivity	35%

des interfaces indirectes. Ce problème s'applique aussi à `URLContentProducer` qui implémente aussi indirectement l'interface `Storable`.

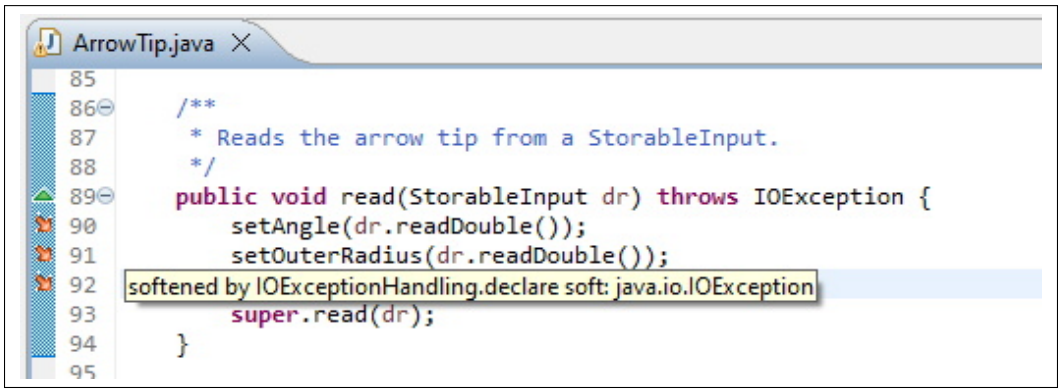

Figure 5.3 Aspect affectant la classe ArrowTip

La classe `EventDispatcher` est une classe statique interne de la classe `AbstractCommand`. `EventDispatcher` gère le patron de conception *observable*. Cette classe n'est pas mise en évidence par les métriques utilisées.

5.3.2 HSQLDB

Le tableau 5.4 montre les 24 classes détectées avec le plus de précision avec les métriques de couplage entrant et sortant, de cohésion et la marche aléatoire pour le système HSQLDB. Comme pour JHotDraw, la première version de NbCluster donne de meilleurs résultats que la seconde. Le couplage sortant, la cohésion et la marche aléatoire détectent des classes complémentaires. La seule classe détectée par le fan-in avec une précision parfaite l'est aussi par la marche aléatoire.

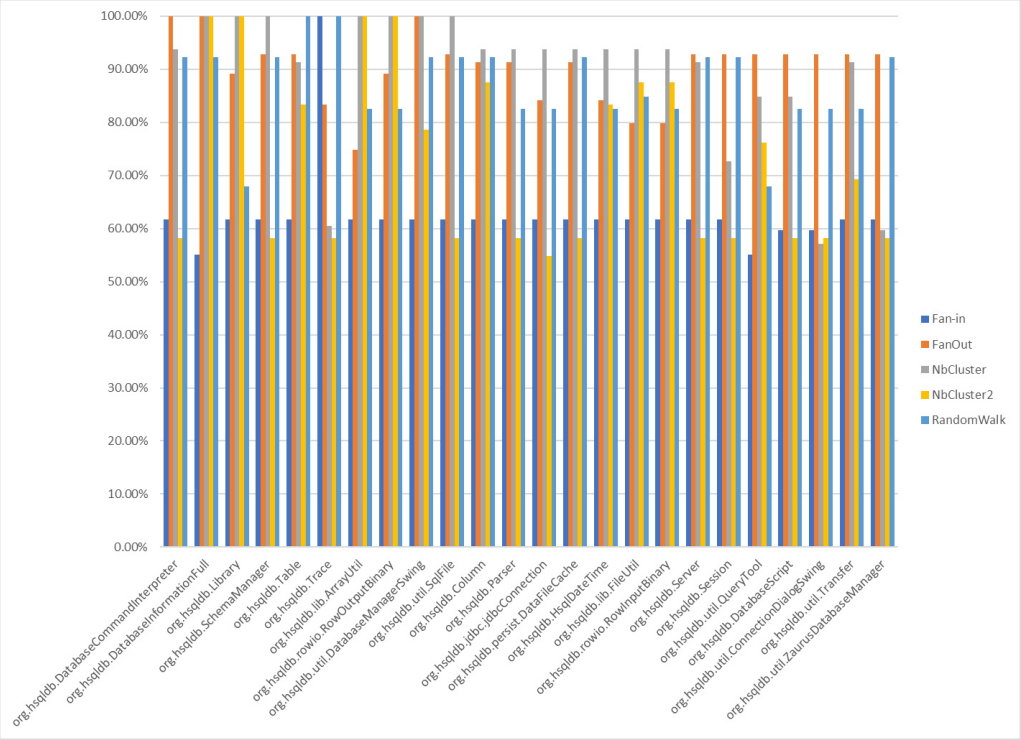

Figure 5.4 Précisions maximales avec fan-in, fan-out, cohésion et Random Walk pour HSQLDB

Le tableau 5.5 montre les précisions maximales en incluant d'autres métriques [1] : LOC (lines of code), NbMethode (nombre de méthodes), CBO (coupling between objects) et la profondeur de hiérarchie. Presque toutes les classes détectées avec une précision de 100% le sont par CBO, LOC ou NbMethode. Seules deux classes sont détectées avec une précision de 100% exclusivement par la profondeur de hiérarchie ou la marche aléatoire. Ces résultats semblent

[1] Les métriques LOC, NbMethode et CBO étaient incluses avec le jeu de données au début du projet.

indiquer que le simple fait qu'une classe soit volumineuse avec beaucoup de méthodes est un bon indicateur de présence de préoccupations transverses. Dans ce système aussi, le couplage entrant et la cohésion (NbCluster) ont une contribution complémentaire, ce qui répond positivement à la QR1. Cependant, les classes détectées avec précision par NbCluster le sont aussi par LOC, CBO ou NbMethode. Il serait intéressant de les inclure dans des travaux futurs. La profondeur de hiérarchie a détecté une classe que les autres métriques n'ont pas détecté aussi précisément.

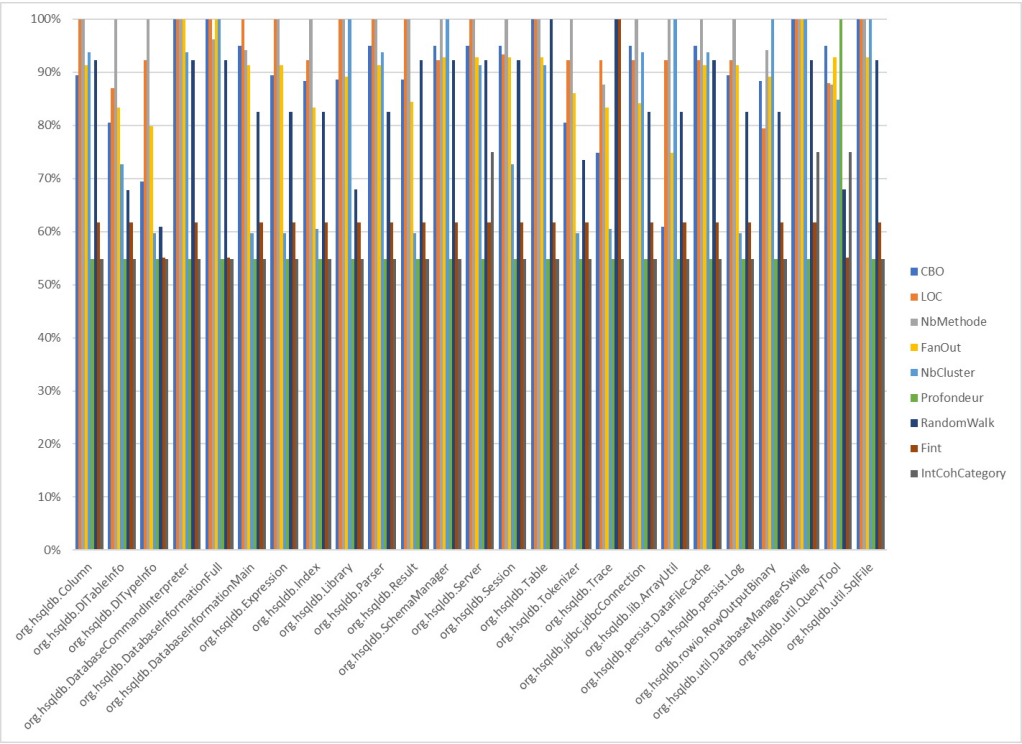

Figure 5.5 Précisions maximales incluant d'autres métriques de classe pour HSQLDB

Le tableau 5.2 montre les classes détectées avec la précision la plus faible. Malheureusement, au moment de rédiger ce mémoire, le plugin AJDT servant à travailler avec les projets AspectJ sur Eclipse a un bogue avec les dernières versions d'Eclipse. Cela empêche d'observer les références croisées dans le projet. Cela ramène un enjeu important d'AspectJ : la compréhensibilité du code source dépend d'un environnement de développement pour mettre en évidence les points de coupure des aspects. Sans cette mise en évidence, il n'y a aucun indice dans les classes si elles sont affectées ou non par un aspect.

La seule option est de comparer le code source de la version aspectualisée et de la version Java. Pour la classe `TableModelHandler` imbriquée dans `TableSorter`, il n'y a pas de différence évidente entre les deux versions. Elle est probablement liée à une référence croisée. Pour la classe `Collector` dans `Expression`, elle ne semble pas avoir été modifiée, mais un aspect vide a été défini à la fin de la classe `Expression`.

Le remaniement aspect de HSQLDB présente des différences importantes par rapport à JHotDraw. Dans JHotDraw, les aspects sont regroupés dans un package de préoccupations transverses. Dans HSQLDB les aspects sont définis à l'intérieur des classes.

Plusieurs des classes moins précisément détectées sont des classes imbriquées. Il est possible que ces classes soient marquées comme porteuses de préoccupation transverse seulement parce que leur classe conteneur l'est aussi.

Tableau 5.2 Quelques classes détectées avec le moins de précision avec HSQLDB

Classe	**Précision maximale**
org.hsqldb.HsqlServerFactory	79%
org.hsqldb.util.PostgresTransferHelper	79%
org.hsqldb.DatabaseURL	78%
org.hsqldb.WebServer	75%
org.hsqldb.jdbc.jdbcDataSourceFactory	75%
org.hsqldb.lib.HsqlTimer.Task	75%
org.hsqldb.persist.DataFileBlockManager	75%
org.hsqldb.util.JDBCTypes	75%
org.hsqldb.Index.IndexRowIterator	75%
org.hsqldb.rowio.RowInputTextQuoted	72%
org.hsqldb.types.JavaObject	71%
org.hsqldb.util.TableSorter.Arrow	71%
org.hsqldb.Expression.Collector	69%
org.hsqldb.util.TableSorter.TableModelHandler	67%

5.4 Conclusion

L'analyse des précisions maximales a permis de confirmer une certaine complémentarité entre le couplage entrant et sortant et la cohésion dans les systèmes JHotDraw et HSQLDB. Cela confirme la QR1. La cohésion des interfaces a aussi été utile avec JHotDraw pour mettre en évidence les interfaces qui représentent une préoccupation transverse. La marche aléatoire a aussi détecté quelques classes complémentaires.

Parmi les classes les plus difficiles à détecter avec précision dans JHotDraw, il y a les classes `Undoable`, qui sont des classes imbriquées prenant en charge une préoccupation transverse dont l'implémentation est spécifique pour chaque sous-classe d'`AbstractCommand`. D'autres classes détectées avec moins de précision sont reliées à la préoccupation de persistance des objets (lecture et écriture), bien que cette préoccupation ne soit pas elle-même complètement remaniée. La mesure de la cohésion des interfaces devrait être modifiée pour tenir compte des interfaces implémentées indirectement. L'analyse du code source de HSQLDB a été limitée par la désuétude de l'outil de développement. Il en ressort néanmoins que pour ce système, les classes plus volumineuses sont plus susceptibles d'être concernées par une préoccupation transverse.

CHAPITRE 6

RÉFLEXIONS SUR LA GESTION DES PRÉOCCUPATIONS TRANSVERSES

La programmation orientée aspect a pour objectif de diminuer la duplication et l'enchevêtrement du code. Cependant, l'utilisation de la programmation orientée aspect peut apporter quelques difficultés.

Ce chapitre a pour but de discuter des enjeux de la prise en charge des préoccupations transverses et de discuter des alternatives. Même s'il y a un déclin de l'intérêt de la communauté pour la programmation orientée aspect, les préoccupations transverses restent un enjeu important. Cette discussion comporte deux volets. Dans un premier temps, nous discutons d'une distinction entre la logique globale et la logique locale d'une préoccupation transverse. Ensuite, nous discutons d'approches alternatives permettant de gérer des préoccupations transverses. Enfin, nous discutons de la gestion de quelques exemples de préoccupations transverses à la lumière des deux premiers volets.

6.1 Logique globale et logique locale

Avant d'observer les alternatives à la programmation orientée aspect, nous proposons une distinction à propos du contenu d'une préoccupation transverse. Par sa nature, une préoccupation transverse implique une certaine complexité qui ne peut être complètement éliminée. À déplacer complètement le code d'une préoccupation transverse à l'extérieur d'une classe, on crée une dépendance implicite qui n'est pas visible à priori dans la classe. Mais cela n'élimine pas le couplage entre la préoccupation transverse et la classe. Y aurait-il une partie de la préoccupation transverse qu'on aurait avantage à garder localement dans les classes ? Ce questionnement nous amène à faire la distinction entre ce que nous appelons la logique globale et la logique locale d'une préoccupation transverse.

La **logique globale** est la partie de la préoccupation transverse qui peut être généralisée. C'est une partie qu'on peut espérer factoriser et centralisée.

La **logique locale** est la partie de la préoccupation transverse qui dépend spécifiquement de la logique de chaque classe où elle est présente. Cette partie nécessite des lignes de code adaptées à chaque classe concernée. Cette partie est plus difficile (voire impossible) à factoriser.

La proportion de la logique globale par rapport à la logique locale est variable d'une préoccupation transverse à l'autre. On peut mettre en question l'intérêt de centraliser la logique locale. Distancer cette partie du code de la classe ne facilite pas forcément la compréhension de la préoccupation transverse ou de la classe. On peut toutefois circonscrire le code de cette logique locale et répertorier les endroits où elle est présente.

Par exemple, pour la préoccupation de la journalisation, la logique globale est assumée par la classe de traçage, appelée parfois *Logger*. L'information générale recueillie peut également faire partie de la logique globale. La logique locale concerne quant à elle les endroits dans le code où une trace doit être générée. S'il faut ajouter une information particulière dans la trace, alors cette information fait partie de logique locale.

L'exemple de la sérialisation consiste à convertir un objet dans un certain format de donnée et de pouvoir reconstruire un objet à partir de données dans ce format. Cela peut servir entre autres à stocker des données dans une mémoire secondaire ou à communiquer des données entre divers programmes. Dans ce cas-ci, la logique globale comprend le format de données converties (JSon, XML, etc.). La logique locale comprend la sélection des attributs à inclure dans la sérialisation ainsi que la méthode de restitution (désérialisation) des objets à partir des données formattées.

La préoccupation *Undo* (défaire), présente dans le système JHotDraw, a été discutée dans des chapitres précédents. La préoccupation *Undo* concerne les sous-classes concrètes de la classe `AbstractCommand`. Une commande concrète doit définir la méthode d'exécution d'une commande, et la préoccupation *Undo* requiert que cette commande définisse aussi une méthode pour défaire cette commande. Cette dernière méthode est définie dans une classe `UndoActivity` nichée dans la commande. Chaque sous-classe concrète de `AbstractCommand` a sa propre classe nichée `UndoActivity`. L'implémentation de cette préoccupation est principalement une logique locale, car la méthode pour défaire la commande dépend de la méthode d'exécution de

la commande. Dans un tel cas, on peut difficilement espérer factoriser cette logique. Même son remaniement en aspect ne la factorise pas. La solution proposée consistait entre autres à créer un aspect pour chaque commande (Marin, 2004). La logique globale comprend l'instanciation et la coordination des objets `UndoActivity`. Cette logique a pu être allégée en termes de lignes de code dans la version aspect.

Les qualités qu'on espère rehausser par une meilleure prise en charge des préoccupations transverse sont la concision (éviter la redondance de la duplication de code), la modularité, la lisibilité, la réutilisabilité (corolaire de la modularité) et la traçabilité.

La concision et la modularité peuvent être améliorées par une factorisation de la logique globale. La lisibilité peut être améliorée par une circonscription du code de la logique locale dans les classes concernées et par une visibilité du couplage avec la logique globale. La concision et la circonscription peuvent améliorer la réutilisabilité du code.

6.2 Approches alternatives

Avant d'aborder des approches alternatives, considérons quelques inconvénients et avantages de la programmation orientée aspect. Un inconvénient est le manque de visibilité locale des points de coupure. Les advices sont comparés à l'instruction `Come From` (Constantinides, Skotiniotis & Störzer, 2004), qui est une inversion de l'instruction `Go To`. Les points de coupure où sont greffés les advices peuvent être mis en évidence par des outils de développement, mais cela rend la lisibilité dépendante de ces outils. Le code source seul devient moins lisible et moins prévisible.

La définition des points de coupure est sensible à la nomenclature des méthodes. Par exemple, un renommage d'une méthode ciblée par un aspect peut rompre l'application de l'aspect à cette méthode. Il s'agit d'une fragilité reconnue dans la littérature (Störzer & Koppen, 2004). De plus, selon l'interaction entre un aspect et une classe, le comportement de la classe peut devenir plus difficile à anticiper.

Centraliser la logique locale en la déplaçant dans un aspect à l'extérieur d'une classe n'aide pas forcément la compréhension de cette logique locale. En reprenant l'exemple de la préoccupation *Undo*, la logique locale de la préoccupation est déduite à partir de la classe de la commande. Mettre le code de la commande côte à côte avec le code de sa préoccupation *Undo* facilite la compréhension du code de *Undo*. Le seul avantage de centraliser cette préoccupation est de répertorier cette préoccupation. Cet avantage pourrait facilement être obtenu en indexant les classes concernées à l'aide d'une annotation.

La programmation orientée aspect apporte quand même des avantages. Les impacts sur la prévisibilité sont moins importants lorsqu'un aspect n'altère pas le résultat du code de la classe ou lorsque le code aspect est temporaire. Par exemple, dans le cas de la journalisation, il s'agit simplement d'extraire de l'information lors de l'exécution. Cela ne change pas le résultat des opérations principales. La programmation orientée aspect peut permettre aussi d'écrire des tests unitaires dont les aspects sont tissés seulement pendant les activités de test. Le programme peut alors faire des vérifications supplémentaires qu'il ne ferait pas en déploiement pour des raisons de performance. On peut inclure ou exclure un aspect sans modifier le code des classes.

6.3 Programmation orientée attribut

La programmation orientée attribut consiste à marquer des éléments du programme pour indiquer une sémantique spécifique (Wada & Suzuki, 2005). Ce marquage prend généralement la forme d'annotations. Une critique affirme que les annotations engendrent de la dispersion du code, qu'elles était supposées modulariser (Steimann, 2006). Cependant, ces annotations sont beaucoup plus faciles à tracer, à référencer. Un outil de développement peut facilement référencer les diverses occurrences d'une annotation. L'annotation réduit donc grandement l'effort nécessaire comparativement à l'ajout d'instructions ou de méthodes.

La sérialisation est un exemple de préoccupation transverse pour laquelle les annotations sont une avenue intéressante. Le choix des attributs à sérialiser fait partie de la logique locale à la classe. Les annotations sont une approche concise pour identifier les éléments à sérialiser.

C'est la logique de la classe qui détermine quels attributs sont sérialisés et comment on peut reconstruire un objet à partir de ces attributs. Souvent, la sérialisation est prise en charge par des méthodes `read()` et `write()`. Si la classe, par les annotations, se limite plutôt à marquer les éléments de cette logique locale, cela laisse une liberté dans le choix du format de sérialisation.

6.4 Fonctionnalités spécifiques des langages de programmation

La déclaration `try-with-resources` est une fonctionnalité en Java permettant d'invoquer automatiquement la méthode `close()` d'une ressource implémentant l'interface `java.lang.AutoCloseable` (Oracle, date inconnue). Il s'agit d'un exemple d'une fonctionnalité d'un langage permettant de gérer une préoccupation transverse spécifique. Dans cet exemple, la logique globale est déjà modularisée dans la méthode `close()`. C'est la prise en charge de la logique locale qui est améliorée par cette fonctionnalité.

Le ramasse-miette gère aussi une préoccupation transverse, celle de la désallocation de la mémoire. Cette fonctionnalité réduit considérablement la complexité de cette préoccupation dans le code source.

6.5 Conclusion

La distinction entre la logique locale et la logique globale d'une préoccupation transverse peut faciliter l'évaluation d'une approche pour la gérer. Nous proposons comme objectif que la logique globale soit factorisée et centralisée, et que la logique locale reste présente localement, mais avec une meilleure concision. Le fait que la logique locale soit visible est important pour que le fonctionnement de la classe reste intelligible. Nous avons aussi abordé quelques approches alternatives à l'orienté aspect. Il y a l'utilisation d'annotations qui permet d'ajouter avec concision de l'information sémantique. Nous avons vu aussi que certaines fonctionnalités de langages de programmation permettent de gérer des préoccupations transverses spécifiques.

CHAPITRE 7

CONCLUSION

La question de recherche principale de ce projet était de savoir si combiner des métriques de classe, en particulier la cohésion et le couplage, pouvait améliorer la détection de préoccupations transverses. Les expérimentations avec l'apprentissage automatisé ont montré une amélioration de la précision, au détriment du rappel. Il en ressort aussi que les contributions des métriques ne sont pas les mêmes d'un système à un autre. Il est donc invraisemblable d'entrainer un algorithme d'apprentissage sur un système pour prédire la présence de préoccupations transverses sur un autre système. L'outil Tanagra, utilisé pour les tests, a une facilité d'utilisation, mais ne permet pas facilement d'automatiser des tests avec de multiples combinaisons de métriques. Pour une meilleure efficience dans de futurs travaux, il serait préférable d'utiliser un langage de programmation comme Python pour automatiser les expérimentations. Cela permettrait d'automatiser les batteries de tests avec plusieurs combinaisons de métriques. Cela permettrait aussi de lister les éléments mal classés pour enquêter sur les préoccupations transverses qui échappent aux métriques utilisées. L'analyse des préoccupations maximales a aussi confirmé une complémentarité entre la cohésion et le couplage entrant.

Étant donné le peu de systèmes remaniés disponibles pour l'étude, et la potentielle incomplétude de leur remaniement aspect, nous nous sommes intéressés à une heuristique pour explorer progressivement le lien entre les métriques et les préoccupations transverses même en sachant qu'elles ne sont pas toutes identifiées. Le principe est de mettre en évidence, pour chaque classe positive, la précision maximale avec laquelle elle est prédite positive par un des algorithmes. En utilisant cette approche avec la classification par seuil de discrimination, facile à implémenter, il appert que plusieurs des métriques étudiées, dont le couplage et la cohésion, détectent avec précision des éléments complémentaires. Combiner ces métriques peut donc améliorer la couverture de précision.

Nous avons discuté au dernier chapitre de différentes approches permettant la gestion de préoccupations transverses. Nous proposons de distinguer, dans le code d'une préoccupation

transverse, ce que nous appelons la logique locale et la logique globale. Nous postulons qu'il est souhaitable que la logique locale d'une préoccupation transverse soit gérée localement et non de façon centralisée. Les remaniement aspects tendent à centraliser non seulement la logique globale, mais aussi la logique locale. Le couplage entre une classe et un aspect est un couplage implicite et invisible sans outil de développement.

Pour des travaux futurs, il serait intéressant d'inclure davantage de métriques à l'étude, entre autres d'autres métriques de cohésion. La cohésion est difficile à bien évaluer avec une seule métrique. Il serait pertinent aussi d'isoler chaque préoccupation transverse pour l'étudier séparément. Il est possible que plusieurs préoccupations transverses soient présentes dans une même classe. Les classificateurs par apprentissage automatisés pourraient aussi être inclus dans l'analyse des précisions maximales. On peut aussi considérer de détecter des préoccupations transverse au niveau de granularité de la méthode plutôt que de la classe. Pour les préoccupations transverses implémentées par des bouts de code dupliqués dans plusieurs méthodes sans appels à d'autres méthodes, le niveau de granularité de la classe pourrait ne pas suffire.

BIBLIOGRAPHIE

Breiman, L. & Cutler, A. (date inconnue). Random Forests [Page web]. Repéré le 2022-08-07 à https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#papers.

Breu, S. & Krinke, J. (2004). *Aspect mining using event traces*. Conference Proceedings présentée à Proceedings. 19th International Conference on Automated Software Engineering, 2004. (pp. 310-315). doi : 10.1109/ASE.2004.1342754.

Breu, S. & Zimmermann, T. (2006). *Mining Aspects from Version History*. Conference Proceedings présentée à 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06) (pp. 221-230). doi : 10.1109/ASE.2006.50.

Bruntink, M., Deursen, A. v., Engelen, R. v. & Tourwe, T. (2005). On the use of clone detection for identifying crosscutting concern code. *IEEE Transactions on Software Engineering*, 31(10), 804-818. doi : 10.1109/TSE.2005.114.

Canfora, G. & Cerulo, L. (2005). *How Crosscutting Concerns Evolve in JHotDraw*. Conference Proceedings présentée à 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05) (pp. 65-73). doi : 10.1109/STEP.2005.13.

Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P. & Tourwé, T. (2006). Applying and combining three different aspect Mining Techniques. *Software Quality Journal*, 14(3), 209-231. doi : 10.1007/s11219-006-9217-3.

Constantinides, C., Skotiniotis, T. & Störzer, M. (2004, septembre). AOP Considered Harmful. *European Interactive Workshop on Aspects in Software*.

Domingos, P. & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *MACHINE LEARNING*, pp. 103–137.

Griswold, W. G., Kato, Y. & Yuan, J. J. (2000). Aspectbrowser : Tool support for managing dispersed aspects. 6.

Halkidi, M., Batistakis, Y. & Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of intelligent information systems*, 17(2), 107-145.

Hannemann, J. & Kiczales, G. (2002). Design pattern implementation in Java and aspectJ. *ACM SIGPLAN Notices*, 37(11), 161-173. doi : 10.1145/583854.582436.

Hosmer, D. W., Lemeshow, S. & Sturdivant, R. X. (2013). *Applied logistic regression* (éd. Third edition /). Hoboken, New Jersey : Wiley. Repéré à http://www.dawsonera.com/depp/reader/protected/external/AbstractView/S9781118548394.

Kersten, G. E., Matwin, S., Noronha, S. J. & Kersten, M. A. (2000). The software for cultures and the cultures in software. *ECIS 2000 Proceedings*, 50. Repéré à https://aisel.aisnet.org/ecis2000/50.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. & Irwin, J. (1997). Aspect-oriented programming. 1241, 220-242. doi : 10.1007/BFb0053381.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. pp. 1137–1143.

Larman, C., Roques, P., Kruchten, P., Burr, E., Baland, M.-C. & Carité, L. (2005). *UML 2 et les design patterns* (éd. 3e édition.). Paris : Pearson Education.

Marin, M., Deursen, A. v. & Moonen, L. (2004). *Identifying aspects using fan-in analysis*. Conference Proceedings présentée à 11th Working Conference on Reverse Engineering (pp. 132-141). doi : 10.1109/WCRE.2004.23.

Marin, M., Moonen, L. & Deursen, A. v. (2005). *A classification of crosscutting concerns*. Conference Proceedings présentée à 21st IEEE International Conference on Software Maintenance (ICSM'05) (pp. 673-676). doi : 10.1109/ICSM.2005.7.

Marin, M., Moonen, L. & Deursen, A. V. (2006). *FINT : Tool Support for Aspect Mining*. Conference Proceedings présentée à 2006 13th Working Conference on Reverse Engineering (pp. 299-300). doi : 10.1109/WCRE.2006.30.

Marin, M., Moonen, L. & Deursen, A. v. (2007). *An Integrated Crosscutting Concern Migration Strategy and its Application to JHOTDRAW*. Conference Proceedings présentée à Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007) (pp. 101-110). doi : 10.1109/SCAM.2007.25.

Marin, M. (2004). *Refactoring jhotdraw's undo concern to aspectj*. Conference Proceedings présentée à Proceedings of the 1st Workshop on Aspect Reverse Engineering (WARE 2004).

Mehrotra, K., Mohan, C. K. & Ranka, S. (1997). *Elements of artificial neural networks*. Cambridge, Mass. : MIT Press.

Oracle. (date inconnue). The try-with-resources Statement [Page web]. Repéré le 2022-08-07 à https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html.

Sadaoui, L., Badri, M. & Badri, L. (2012). Improving Class Cohesion Measurement : Towards a Novel Approach Using Hierarchical Clustering. *Journal of Software Engineering and Applications*, 05(07), 449-458. doi : 10.4236/jsea.2012.57051.

Schölkopf, B., Smola, A. J. & Bach, F. (2002). *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT press.

Steimann, F. (2006). The Paradoxical Success of Aspect-Oriented Programming. *ACM SIGPLAN NOTICES*, 481–497.

Störzer, M. & Koppen, C. (2004, septembre). PCDiff : Attacking the Fragile Pointcut Problem, Abstract. *European Interactive Workshop on Aspects in Software*.

Störzer, M., Eibauer, U. & Schoeffmann, S. (2006). Aspect mining for aspect refactoring : An experience report. *Towards Evaluation of Aspect Mining—TEAM 2006—*, 17.

Tonella, P. & Ceccato, M. (2004). *Aspect mining through the formal concept analysis of execution traces*. Conference Proceedings présentée à 11th Working Conference on Reverse Engineering (pp. 112-121). doi : 10.1109/WCRE.2004.13.

Van Deursen, A., Marin, M. & Moonen, L. (2005). *AJHotDraw : A showcase for refactoring to aspects*. Conference Proceedings présentée à Workshop on Linking Aspect Technology and Evolution.

Wada, H. & Suzuki, J. (2005). An Introduction to Attribute-Oriented Programming [Web Page]. Repéré à https://web.archive.org/web/20050526230023/http://dssg.cs.umb.edu/resources/attribute-oriented-programming.html.

Wilson, D. R. & Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of artificial intelligence research*, 6, 1–34.

Zhang, C. & Jacobsen, H. (2012). Mining Crosscutting Concerns through Random Walks. *IEEE Transactions on Software Engineering*, 38(5), 1123-1137. doi : 10.1109/TSE.2011.83.