Prédiction de la stabilité des interfaces de classes Java : une approche par analogie

David Grosser — Houari A. Sahraoui — Petko Valtchev

DIRO, Université of Montréal,
CP 6128, succ Centre-ville,
Montréal QC H3C 3J7, Canada
{*grosserd, sahraouh, valtchev*}*@iro.umontreal.ca*

RÉSUMÉ. *Prédire la stabilité d'un logiciel orienté-objet (OO), i.e. comment peut-il évoluer tout en préservant sa conception ? est un facteur clé pour la maintenance du logiciel. S'il est bien conçu, il doit être capable d'évoluer tout en conservant une compatibilité entre versions. La stabilité, comme la plupart des facteurs de qualité, est un phénomène complexe et pouvoir la prédire est un véritable enjeu d'actualité. Nous présentons une nouvelle approche fondée sur le paradigme du Raisonnement à Partir de Cas (RàPC). Afin de prédire les chances qu'un élément logiciel puisse briser la compatibilité ascendante, notre méthode utilise la connaissance de logiciels pour lesquels plusieurs versions sont disponibles. Une base de cas est construite à partir d'un ensemble de métriques structurelles mesurées sur chaque version. La stabilité des nouveaux composants est calculée à l'aide d'une mesure de proximité. Les résultats des analyses conduites par cette méthode sur un large ensemble de données, sont comparés à la méthode d'apprentissage inductif classique des arbres de décision.*

ABSTRACT. *Predicting stability in object-oriented (OO) software, i.e., the ease with which a software item can evolve while preserving its design, is a key feature for software maintenance. In fact, a well designed OO software must be able to evolve without violating the compatibility among versions. Stability, like most quality factors, is a complex phenomenon and its prediction is a real challenge. In this paper, we present a novel approach which relies on the case-based reasoning (CBR) paradigm. Thus, to predict the chances of an OO software item to break downward compatibility, our method uses knowledge of past evolution extracted from different software versions. New components are assigned a stability value based on their degree of proximity to known cases from the base. A comparison of our similarity-based approach to a classical inductive method such as decision trees, is presented which included various tests on large datasets from existing software.*

MOTS-CLÉS : *Maintenance du logiciel, prédiction de la stabilité, métriques structurelles, classification, raisonnement à partir de cas, mesure de similarité.*

KEYWORDS: *software maintenance, stability prediction, software metrics, classification, case-based reasoning, similarity.*

RSTI - L'objet – 9/2003. LMO'2003, pages 61 à 73

1. Introduction

Aujourd'hui, nous pouvons affirmer que la programmation par les objets a atteint un stade de maturité avancé, au regard de la quantité considérable d'applications orienté-objets utilisées et de la variété des langages disponibles fondés sur ce paradigme. De plus, la plupart des packages existants sont passés par de multiples révisions, produisant à chaque fois de nouvelles versions plus achevées. Cependant, l'écriture de nouvelles versions est une tâche difficile en temps et en effort, du fait de la complexité croissante des applications. Pressman estime par exemple que plus de 60% des ressources consacrées au développement du logiciel sont dédiées à la maintenance [PRE 97]. De ce pourcentage, 80% concerne directement ou indirectement l'évolution perfective ou adaptative du logiciel [PIG 97].

Même si la programmation par les objets présente des particularités qui facilitent l'évolution, elle n'est toutefois pas exempte de ce phénomène. En particulier, comme les systèmes à objets sont de plus en plus ouverts, il est important que le passage d'une version à une autre préserve la compatibilité ascendante de l'interface des classes et donc la stabilité de leur conception. Il est donc nécessaire de développer des outils qui permettent d'évaluer la stabilité d'éléments logiciels à travers la détection de situations symptomatiques d'instabilité future. De façon générale, nous définissons la stabilité d'un élément logiciel comme la capacité à pouvoir évoluer tout en préservant la compatibilité avec les autres entités du système.

La prédiction de la stabilité est un problème complexe compte tenu de la variabilité intrinsèque des applications. La pratique générale consiste à trouver par induction les caractéristiques des situations symptomatiques en se basant sur les données historiques disponibles. Malheureusement, dans le domaine du logiciel et contrairement à certaines disciplines comme la sociologie ou la médecine, les chercheurs ne disposent pas de banques de données suffisantes à partir desquelles des échantillons représentatifs peuvent être sélectionnés. En conséquence, les modèles existants aujourd'hui sont difficiles à généraliser et à réutiliser.

Pour contourner ce problème, nous proposons dans cet article une approche basée sur le principe de comparaison par similarité. Elle consiste dans notre contexte à prédire/évaluer une caractéristique de qualité (stabilité) pour un élément logiciel (classe) en le comparant à d'autres éléments pour lesquelles nous connaissant déjà cette caractéristique. Nous donnons dans un premier temps un aperçu général des travaux connexes (section 2). Dans la section 3, nous définissons la notion de stabilité des interfaces de classes Java. Notre approche fondée sur le raisonnement à partir des cas (RàPC) sera décrite dans la section 4.1 et évaluée dans la section 5.

2. Prédiction de la qualité des logiciels à objets

La majorité des caractéristiques de qualité des logiciel ne sont pas directement mesurables *a priori* (fiabilité, maintenabilité, réutilisabilité, etc.). Ce constat a poussé la communauté du génie logiciel à explorer - souvent de manière empirique - les

possibilités de prédire ces caractéristiques à partir d'attributs mesurables du logiciel (couplage, cohésion, taille, etc.) [FEN 00]. Les premiers travaux qui ont porté sur la construction de modèle prédictifs se sont fondés sur des techniques classiques bien connues en statistiques, comme la régression linéaire des moindres carrés ou les régressions robustes (une étude comparative de ces travaux est présentée dans [BRI 02]). Par la suite, de nombreux chercheurs ont utilisé des techniques alternatives, inspirées pour la plupart de l'apprentissage automatique. Ainsi, dès le début des années 90, des modèles prédictifs de la fiabilité [KAR 92], de la taille [HAK 93] ainsi que de l'effort de développement [WIT 94, WIT 95] ont été proposés. Malgré la performance relative de ces modèles, leur nature de type "boîte noire", a limité fortement leur utilisation.

Plus récemment, d'autres techniques ont été appliquées en génie logiciel, telles que les arbres de décision [MAO 98]. Ces techniques se limitent essentiellement à un problème de classification et introduisent un biais d'apprentissage lié au choix des valeurs seuils de décision trop spécifiques aux échantillons utilisés. Par la suite, la logique floue a été explorée comme moyen de gérer l'incertitude inhérente à la nature du logiciel et de contourner la problématique des valeurs seuils [GEN 00, SAH 01] sans pour autant résoudre le problème de la non représentativité des échantillons.

3. Problème de la stabilité

Pendant sa durée d'opération, un logiciel subit de nombreux changements. Ces changements sont dus à des corrections d'erreurs, des adaptations à de nouvelles technologies et surtout à la prise en compte de nouveaux besoins. En général, ces changements ont pour effet, à plus ou moins long terme, de dégrader le logiciel au point de le rendre imprévisible [PAR 94]. Les logiciels qui sont destinés à une utilisation de longue durée doivent donc pouvoir supporter l'évolution. Malgré la prise de conscience de ce problème par la communauté du génie logiciel, les techniques proposées aujourd'hui ne garantissent pas cette stabilité [FAY 02].

Pour faire face à ce problème, de nombreux travaux ont vu le jour. Ainsi, [MAR 97] propose des règles de conception basées sur la gestion des dépendances et l'abstraction pour garantir la stabilité des grands systèmes. Dans le même ordre d'idée, [FAY 02] propose un model permettant de concentrer les efforts de conception sur un noyau stable du système (*Enduring Business Themes et Business Objects*) et de laisser une certaine liberté pour une partie périphérique appelée à subir de nombreux changement (*Industrial Objects*). Ces deux travaux ne proposent pas de méthodes permettant d'évaluer la stabilité. Il est donc très difficile d'évaluer concrètement leur impact sur cette caractéristique. D'autres travaux ont ciblé spécifiquement l'évaluation de la stabilité [DEM 99, BAN 00] dans le cas particulier des frameworks.

Pour notre part, nous nous intéressons à la stabilité du logiciel à objets en général. Toutefois, nous étudions celle-ci au niveau des classes tout en considérant les dépendances entre classes. Ainsi, nous considérons qu'une classe est stable si son interface

publique demeure stable d'une version à l'autre. Si l'on considère la classe c_i (i indiquant la version du système), l'interface publique $I(c_i)$ représente l'ensemble des méthodes locales ou héritées, que toute instance d'une classe différente de c_i peut appeler sur une instance de c_i. En Java par exemple, les méthodes *public*, *protected* et de portée le *package* sont considérées. Le niveau la stabilité $NS(c_i \rightarrow c_{i+1})$ entre deux versions i et $i+1$ est défini comme le pourcentage de $I(c_i)$ qui est inclus dans $I(c_{i+1})$.

4. Prédiction par analogie

Le principe général de raisonnement associé au RàPC consiste à résoudre de nouveaux problèmes par *recherche* et *adaptation* des solutions associées à des problèmes similaires survenus par le passé. Ces problèmes sont représentés et stockés sous forme de cas individuels dans une mémoire appelée *base de cas* [AAM 94]. L'utilisation d'un historique de cas permet de réduire de manière conséquente la nécessité d'analyser en profondeur le problème à résoudre, car les solutions des situations passées peuvent être réutilisées, moyennant adaptation [CUN 99]. Le RàPC s'applique bien à des problèmes pour lesquels aucun modèle formel n'est connu et les interactions entre les différents paramètres mal comprises. L'espoir réside dans le fait que la modélisation fines des interactions peut être substituée par réutilisation des problèmes précédemment résolus, stockés dans les cas.

Le choix des cas les plus proches utilisés pour construire la solution du nouveau cas est guidé par un raisonnement par analogie fondé sur la description des cas de la base. Cette partie descriptive est généralement modélisée par un ensemble d'attributs. Les analogies sont détectées par un mécanisme de correspondance, typiquement une mesure de comparaison.

Dans le cas simple où la solution du problème est modélisé par une seule variable dépendante, le RàPC peut être considéré comme une sorte de classifieur, qui pour une description donnée déduit une valeur pour la variable ciblé. Dans ce cadre d'application particulier, le RàPC peut-être considéré comme une méthode d'apprentissage automatique, telle que l'apprentissage par les exemples (*instance-based learning*).

4.1. *RàPC et prédiction de la stabilité*

L'absence de modèles prédictifs disponibles sur l'évolution du logiciel nous a conduit à adopter une stratégie de type RàPC, en émettant l'hypothèse que deux éléments logiciels qui présentent des caractéristiques similaires vont évoluer de manière identique. Dans notre contexte, les caractéristiques d'un élément logiciel (sa description) vont être représentées par un ensemble de *métriques structurelles* et sa stabilité par une variable booléenne à valeur dans *stable* ou *instable*. Si l'on considère l'évolution de la stabilité entre deux versions i et $i+1$ d'un système à objet, les

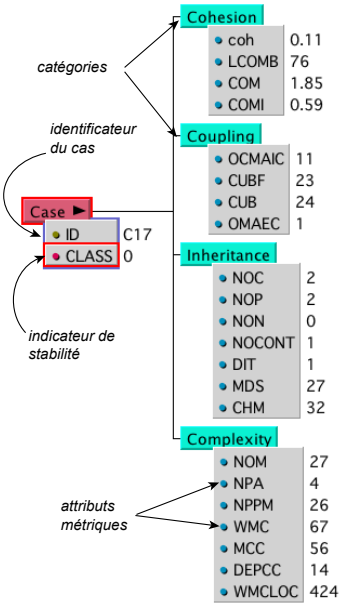
métriques logicielles sont extraites depuis la version i, alors que la mesure de stabilité est dérivée d'une analyse différentielle entre les versions i et $i+1$.

Nous présentons ici trois aspects importants de la méthode : la représentation des cas (§4.2), la méthode de recherche des cas pertinents (§4.3) et la mesure de similarité utilisée (§4.4).

4.2. *Représentation des cas*

Le problème de la représentation des cas en RàPC consiste principalement à choisir un ensemble d'attributs pour décrire les cas, puis à trouver une structure appropriée pour la représentation de ces attributs et enfin à décider de l'organisation et de l'indexation de la base de cas. Ce dernier aspect conditionne particulièrement l'efficacité de la phase de recherche.

Pour cette étude de la stabilité du logiciel, nous avons utilisé le système $\mathcal{IKBS}$ (*Iterative Knowledge Based System*), développé pour la représentation et le traitement de connaissances en systématique [GRO 02]. Cet outil permet de représenter des cas sous une forme structurée, en définissant au préalable un modèle. Il dispose également de différentes méthodes de raisonnements dont une méthode de RàPC et de construction d'arbres de décision [CON 99], que nous avons adaptées aux besoins de notre étude.



Symbole	Nom
Métriques de cohésion	
LCOM	lack of cohesion methods
COH	cohesion
COM	cohesion metric
COMI	cohesion metric inverse
Métriques de couplage	
OCMAIC	other class method attribute import coupling
OCMAEC	other class method attribute export coupling
CUB	number of classes used by a class
CUBF	number of classes used by a memb. funct.
Métriques d'héritage	
NOC	number of children
NOP	number of parents
NON	number of nested classes
NOCONT	number of containing classes
DIT	depth of inheritance
MDS	message domain size
CHM	class hierarchy metric
Métriques de complexité en taille	
NOM	number of methods
WMC	weighted methods per class
WMCLOC	LOC weighted methods per class
MCC	McCabe's complexity weighted meth. per class
DEPCC	operation access metric
NPPM	number of public and protected meth. in a class
NPA	number of public attributes

Figure 1. *Exemple de représentation structurée d'un élément logiciel à l'aide de métriques et description des métriques utilisées*

Un exemple de représentation structurée d'un cas est donné à la figure 1. Dans cette représentation, un cas est le résultat de l'analyse de l'évolution d'un élément logiciel entre deux versions d'un même logiciel. Chaque cas est décrit dans une représentation objet-attribut-valeur, où les objets sont des groupes de métriques structurelles — cohésion, complexité, etc. — les attributs associés aux objets correspondent aux différentes métriques utilisées et les valeurs associées aux attributs sont le résultat du calcul de chaque métrique pour l'élément logiciel considéré. Chaque cas est identifié par un identificateur unique (ID sur la figure 1). La représentation inclut un attribut explicatif booléen, dont la valeur correspond à l'indice de stabilité de l'élément logiciel considéré (1 dénote un élément logiciel stable, 0 un élément instable).

Dans l'exemple, le cas $n^{\circ}C17$ est une classe qualifiée d'instable, possède 27 méthodes (cf. attribut NOM pour *Number Of Methods*, du groupe complexité), la mesure de cohésion est 1.85 (groupe Cohesion), etc.

4.3. *Recherche des cas les plus proches*

L'objectif est de trouver un sous-ensemble de cas connus qui correspondent au mieux à la situation décrite par un cas donné, cible du raisonnement. Cet ensemble est construit à l'aide d'une mesure de comparaison (ou mesure de similarité), et ses éléments sont appelés "cas les plus proches" du fait de leur proximité élevée par rapport au nouveau cas, dans l'espace de description. La phase de recherche consiste donc à construire l'ensemble des cas les plus proches, à partir de la description d'un problème qui peut être partielle.

L'algorithme parcourt la base de cas de manière exhaustive. Chaque cas C_i de la base est comparé au nouveau cas C à l'aide d'une mesure de similarité (cf. § 4.4) calculée sur la base des métriques structurelles. En fonction de la valeur de la mesure, le cas C_i est inséré ou non dans l'ensemble des cas les plus proches. L'algorithme est de complexité linéaire, mais peut encore être amélioré par différente techniques d'optimisation. Voir [WIL 97] pour un éventail des techniques utilisées dans le cadre du RàPC.

Afin d'augmenter les chances d'une prédiction correcte, le nombre d'éléments de l'ensemble peut être d'une taille $k > 1$, k étant un paramètre de l'algorithme. La technique de recherche étant connue sous le nom de *k-plus-proche-voisins* (*k-NN*). Dans l'étude suivante, nous avons choisi la stratégie la plus simple en positionant $k = 1$.

4.4. *Mesure de similarité*

Par la suite, nous considérons qu'un ensemble d'éléments logiciels dont l'évolution de la stabilité est connue, sont stockés dans une base de cas. Chaque cas est décrit par un point dans le *n-espace-euclidien* où les coordonnées sont les métriques structurelles. De nombreuses mesures de similarités ont été proposées dans la littérature,

incluant des variations de distance de Minkowski, comme la distance euclidienne, la distance de Manhattan, etc. Dans notre étude, la distance utilisée est dérivée de la distance de Manhattan. Elle est définie à deux niveaux : niveau métrique (local), et niveau élément logiciel (global).

Pour chaque variable A_i, le facteur de similarité entre deux exemples x et y, noté $sim(x.A_i, y.A_i)$, est défini comme la différence absolue entre les deux valeurs de A_i, normalisée par l'étendue du domaine de A_i. Formellement, $sim(x.A_i, y.A_i)$ est calculée de la manière suivante :

$$sim(x.A_i, y.A_i) = 1 - \frac{|x.A_i - y.A_i|}{|\mathrm{dom}(A_i)|} \quad (1)$$

où $|\mathrm{dom}(A_i)|$ correspond à la différence maximale entre les deux extremum de A_i. Le facteur local est ainsi clairement tel que $\forall x, y \in D^2, sim(x.A_i, y.A_i) \in$ [0 1].

Les contributions relatives de la mesure locale appliquée à chaque variable A_i sont combinées en une unique valeur caractérisant la similarité globale entre deux cas. Nous utilisons pour cela une combinaison linéaire des mesures locales, telle que :

$$Sim(x, y) = \frac{\sum_{i=1}^{i=p} \beta_i \, sim(x.A_i, y.A_i)}{p} \quad (2)$$

où $\beta_i \geq 0$ est le poid associé à la métrique A_i.

Dans l'étude exposée à la section suivante, nous considérons initialement des poids égaux à 1 : $\forall i \in \{0, \ldots, p\}, \beta_i = 1$. Ce choix reflète le manque de connaissance concernant la contribution respective de chaque mesure pour la prédiction de la stabilité. Un des objectif de cette étude consiste justement à préciser le rôle de chaque métrique d'un point de vue de la stabilité d'un élément logiciel, par une approche expérimentale.

5. Évaluation

Pour évaluer notre méthode de prédiction basée sur le RàPC, nous l'avons appliqué à la prévision de la stabilité d'un ensemble de classes d'applications développés en Java. 691 classes ont été sélectionnées parmis deux systèmes (Jedit et Jetty) pour lesquels nous disposions d'au moins deux versions différentes, puis analysées à l'aide de l'outil ACCESS de l'environnement Discover© pour l'extraction des métriques[1].

Les résultats sont comparés à ceux obtenus par l'application d'un modèle de prédiction basé sur les arbres de décision. Pour estimer la précision respective des deux classifieurs, deux techniques classiques de validation ont été employées : la validation croisée avec dix groupes (*10-fold-cross-validation*) et la validation par exclusion

1. Environnement disponible à l'URL *http ://www.mks.com/products/discover/developer.shtml*.

d'un élément (*Leave-one-out*). Les deux méthodes permettent d'estimer l'erreur de généralisation basée sur le ré-echantillonage [WEI 91, SHA 93].

Dans les expériences suivantes, une classe c est dite **stable**, si son interface publique dans la version i est incluse dans son interface publique dans la version $i+1$. Sinon, c est **instable**. Puisque la structure d'un composant logiciel joue un rôle prépondérant dans la détermination de sa stabilité, comme variables pour la description des cas de la base, 22 métriques structurelles ont été choisies puis partitionnées en quatre catégories : *couplage*, *cohésion*, *héritage* et *complexité* (voir figure 1).

Pour chaque méthode de validation et pour chaque algorithme d'apprentissage, quatre fonctions d'estimation de précision sont calculées :

– Degré de correction (*correctness*) : pourcentage absolu des prédictions correctes,

– J(instable) : pourcentage absolu des prédictions d'instabilité correctes,

– J(stable) : pourcentage absolu des prédictions de stabilité correctes,

– J-index : (le *J-index* de Youden) le degré moyen de correction (par classe),

Le degré de correction est défini par :

$$Correctness = \frac{\sum_{i=1}^{k} n_{ii}}{\sum_{i=1}^{k} \sum_{j=1}^{k} n_{ij}} \tag{3}$$

où n_{ij} est le nombre des cas d'entrainement réellement étiquettés par C_i mais classifiés comme C_j. En particulier, n_{ii} est le nombre des membres de la classe C_i qui ont été correctement classifiés.

Les données sur la qualité du logiciel sont souvent *deséquilibrés*, c'est-à-dire que certaines classes (étiquettes) sont beaucoup plus fréquentes que d'autres. Pour donner plus de poids aux données ayant des étiquettes minoritaires, nous avons utilisé l'index *J-index de Youden* [YOU 61]. Le J-index est la moyenne des degrés de correction par étiquette. Il mesure la précision supposant que la probabilité *a-priori* de chaque étiquette est la même :

$$J\text{-}index = \frac{1}{k} \sum_{i=1}^{k} \frac{n_{ii}}{\sum_{j=1}^{k} n_{ij}} \tag{4}$$

5.1. *Comparaison RàPC vs arbres de décision*

La première expérience consiste à classifier les différentes classes en considérant les valeurs respectives des 22 métriques. Le Tableau 1 montre les résultats de l'estimation de la précision entre le RàPC et les arbres de décision. Manifestement, RàPC donne de meilleurs résultats pour les deux critères d'évaluation, la correction et le J-index, et pour les deux méthodes de validation. Pour la validation croisée à 10 groupes, le J-index départage les deux méthodes d'une manière significative car il est de 5%

plus élevé avec le RàPC qu'avec les arbres de décision. Cette différence est encore plus grande avec le critère correction (6%).

Méthode d'apprentissage	"Leave One Out" validation			
	J-index	J(unstable)	J(stable)	Correctness
CBR	71,07%	60,17%	81,98%	74,53%
Arbres de décision	68,40%	68,22%	68,57%	68,45%
	10-fold cross-validation			
CBR	70,15%	59,45%	80,85%	73,33%
Arbres de décision	65,05%	64,98%	65,12%	65,27%

Tableau 1. *Estimations de la précision du RàPC et des arbres de décision*

Ces résultats relativement faibles suggèrent que le problème de la prédiction du facteur stabilité du logiciel, est un problème difficile. Nous croyons cependant que notre approche peut donner des résultats sensiblement meilleurs avec des données plus nombreuses et moins ambigues. D'autre part, nous émettons l'hypothèse que les résultats peuvent être améliorés par un choix mieux approprié des métriques utilisées : le choix arbitraire des 22 métriques n'étant pas nécessairement pertinent pour la prédiction de la stabilité. La prochaine expérience pose les fondements d'un choix plus subtile des sous-ensembles de métriques qui pourraient augmenter la précision des résultats.

5.2. *Réduction de l'ensemble des métriques utilisées*

Dans cette seconde expérience, nous avons cherché à améliorer les résultats par une méthode de sélection d'un sous-ensemble pertinent de métriques. Dans ce but, nous analysons dans un premier temps l'impact de chacune des métriques prises séparemment sur la stabilité. La colonne rang du Tableau 2 présente le classement obtenu selon le J-index.

L'observation du J-index permet d'apprécier la différence significative entre les meilleurs (60% pour CHM) et les plus faibles taux (49% pour OMAEC). Par exemple, ce résultat laisse entendre que la *Class Hierarchy Metric* (CHM) est fortement corrélée avec la stabilité, ce qui est intuitivement reflétée par le fait que la chance pour qu'une classe particulière devienne *instable* est plus élevée lorsque la classe appartient à une grande hiérarchie (entre 50 et 354 classes dans notre expérience).

De même, le *nombre des méthodes* (NOM, NPA et WMC) semble être corrélé avec la stabilité. Cette corrélation se manifeste sous une forme négative, le facteur J(instable) d'estimation de l'instabilité pour cette métrique étant relativement élevé comparé au même facteur pour les autres métriques. Au contraire, la métrique mesurant le nombre des méthodes/attributs exportant du couplage *Other Class Method or*

Métriques		Validation "Leave One Out"				
Catégorie	**Symbole**	**J-index**	**J(instable)**	**J(stable)**	**Précision**	**Rang**
Cohésion		**58,30%**	**38,14%**	**78,46%**	**64,69%**	
	COH	54,46%	31,78%	77,14%	61,65%	12
	LCOMB	51,42%	6,36%	96,48%	65,70%	18
	COM	55,25%	19,07%	91,43%	66,71%	10
	COMI	56,18%	22,03%	90,33%	67,00%	8
Couplage		**59,38%**	**44,92%**	**73,85%**	**63,97%**	
	OCMAIC	52,52%	23,73%	81,32%	61,65%	15
	CUBF	53,76%	28,39%	79,12%	61,79%	14
	CUB	55,25%	30,93%	79,56%	62,95%	11
	OCMAEC	49,53%	7,63%	91,43%	62,81%	22
Héritage		**63,71%**	**42,37%**	**85,05%**	**70,48%**	
	NOC	51,11%	5,08%	97,14%	65,70%	19
	NOP	49,56%	0,00%	99,12%	65,27%	21
	NON	51,50%	13,98%	89,01%	63,39%	17
	NOCONT	50,00%	0,00%	100,00%	65,85%	20
	DIT	52,01%	22,03%	81,98%	61,51%	16
	MDS	58,90%	35,59%	82,20%	66,28%	3
	CHM	60,30%	42,80%	77,80%	65,85%	1
Complexité		**57,25%**	**41,53%**	**72,97%**	**62,23%**	
	NOM	57,69%	35,17%	80,22%	64,83%	4
	NPA	57,24%	24,15%	90,33%	67,73%	5
	NPPM	56,02%	25,00%	87,03%	65,85%	9
	WMC	59,30%	37,29%	81,32%	66,28%	2
	MCC	56,75%	32,63%	80,88%	64,40%	6
	DEPCC	53,98%	22,46%	85,49%	63,97%	13
	WMCLOC	56,75%	32,63%	80,88%	59,62%	7
Toutes les métriques		**71,07%**	**60,17%**	**81,98%**	**74,53%**	

Tableau 2. *Validation par "exclusion d'un élément" de l'approche RàPC*

Attribute Exporting Coupling (OCMAEC) semble n'avoir aucun impact sur la stabilité de la classe.

La second étape consiste à extraire le sous-ensemble pertinent. Une série d'essai est effectué commençant par l'ensemble singleton composé de la meilleure métrique (CHM). à chaque nouvel essai, l'ensemble de métriques est augmenté d'un nouvelle métrique en suivant le rang. Ainsi, le quatrième essai de la série est effectué sur l'ensemble de métriques : CHM, WMC, MDS, et NOM. Notre but est de trouver la combinaison optimale de métriques sans fixer sa taille *a priori*.

Le diagramme résultant (Figure 2) indique que bien qu'une métrique particulière puisse avoir une capacité prédictive élevée, elle peut néanmoins influer de manière neutre ou voire même négative sur la performance conjointe d'un groupe de métriques. Par exemple, COH qui est rangé 12e dans l'ordre global, une fois rajoutée aux 11 meilleures métriques, détériore de manière significative l'exactitude de la prédiction.

Cinq métriques ont un impact négatif sur la précision de la méthode : COMI, NPPM, CUBF, COH and WMC. Les deux dernières se sont révélées particulièrement inadaptées. Ces métriques "non pertinentes" sont représentées par des barres en noir sur le diagramme.

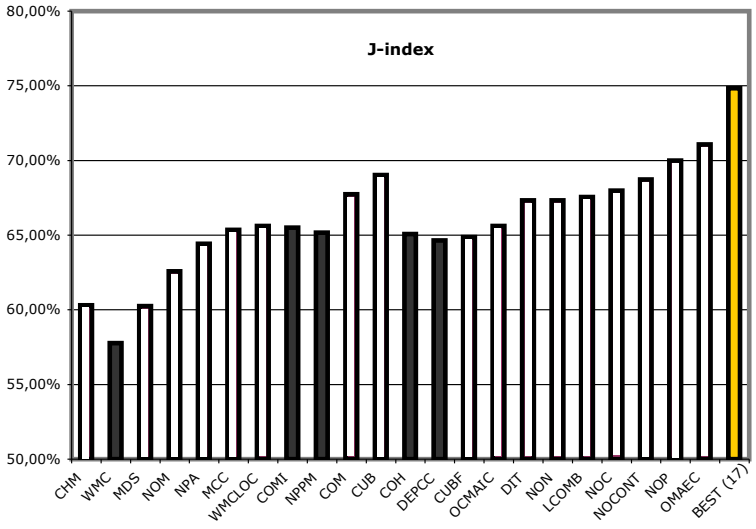


Figure 2. *Contribution relative de chaque métrique*

Pour évaluer l'effet de la mise-à-l'écart de ces cinq métriques, nous avons effectué une nouvelle tâche de classification sur les 17 métriques restantes. Le score obtenu avec le RàPC est représenté par la dernière barre du graphique.

Les résultats obtenus pour le RàPC et les arbres de décisions avec les 17 métriques restantes sont illustrés sur la Figure 3. Comparativement aux résultats de la première expérience, on peut observer une amélioration d'environ 3.5% pour RàPC et 2.5% pour les arbres de décision. Dans le contexte difficile de la prévision de stabilité de logiciel, cette amélioration est significative. D'une manière pragmatique, cela signifie que nous pouvons prévoir la stabilité d'une classe avec une confiance de 75%, en ne considérant que 17 sur les 22 propriétés structurales dans l'ensemble initial. Il sera certainement difficile de pousser bien plus loin ce seuil dans notre cadre expérimental actuel, particulièrement parce que beaucoup de facteurs influençant la stabilité (spécificité du domaine d'application, modèle de programmation, etc...) ne sont pas explicités.

Méthode d'apprentissage	"Leave One Out" validation			
	J-index	J(unstable)	J(stable)	Correctness
CBR	75,17%	64,41%	85,93%	78,58%
Arbres de décision	68,71%	68,36%	69,04%	71,27%
	10-fold-cross-validation			
CBR	73,51%	62,27%	84,74%	76,13%
Arbres de décision	67,58%	67,11%	68,05%	69,98%

Tableau 3. *Estimation de la précision avec les 17 meilleures métriques*

6. Conclusion

Dans cet article, nous proposons une approche du type RàPC pour la prédiction de la stabilité des éléments logiciels, sur la base d'un ensemble de métriques logicielles. La technique présentée considère que chaque métrique est une coordonnée dans un espace multidimensionnel, une dimension par métrique, sur lequel une fonction de similarité est appliquée. La stabilité de chaque nouvel item est calculé en fonction du cas le plus proche trouvé dans la base de cas.

Le modèle prédictif obtenu donne de bons résultats, si l'on considère d'une part, que les données sont en nombre insuffisant et de faible représentativité étant donné la diversité des logiciels orienté-objets développés à ce jour et d'autre part qu'aucun modèle validé n'existe pour ce type de problème. De plus, le modèle de stabilité utilisé est un modèle rudimentaire qui ne tient pas compte des nouveaux besoins pris en compte entre les deux versions.

La stratégie de prédiction de la stabilité fondée sur la similarité permet de plus d'éviter le problème de sur-généralisation des modèles de classification usuels. Les résultats préliminaires que nous exposons montrent qu'une statégie de type RàPC très simple - 1 cas le plus proche utilisé, poids des métriques identiques, pas de théorie du domaine - peut donner des résultats significativement meilleurs qu'un arbre de décision construit à partir de la même base de cas.

L'objectif de nos futures recherches concerne une application de la technique utilisée avec un nombre variable de voisins ($k > 1$) intervenant dans la prise de décision finale. La fine pondération des métriques est un autre aspect clef du problème car, comme le montre les résultats de cette expérience, des poids appropriés permettent de réduire de manière conséquente le bruit généré par les métriques les moins significatives. De plus, nous souhaitons introduire certaines connaissances supplémentaires, telles que des connaissances sur la structure des éléments logiciels manipulés (la hiérarchie de classe étudiée, ses interactions avec les autres classes, etc.) dans les algorithmes utilisés. Finalement, nous allons reconduire l'expérience en considérant un modèle plus sophistiqué de la stabilité tel que suggéré dans la section 3.

7. Bibliographie

[AAM 94] AAMODT A., PLAZA E., Case-Based Reasoning : Foundational Issues, Methodological Variations, and System Approaches , *AI Communications*, vol. 1, n° 7, 1994, p. 39-59.

[BAN 00] BANSIYA J., Evaluating Framework Architecture Structural Stability , *ACM Computing Surveys (CSUR)*, vol. 32, 2000.

[BRI 02] BRIAND L., WUST J., Empirical studies of quality models in object-oriented systems , PRESS A., Ed., *Advances in Computers*, 2002.

[CON 99] CONRUYT N., GROSSER D., Managing complex knowledge in natural sciences , K.D. ALTHOFF R. BERGMANN L. B. E., Ed., *Proc. 3rd International Conference on Case-Based Reasoning (ICCBR-99)*, LNCS subseries, Springer-Verlag, 1650, 1999, p. 401-414.

[CUN 99] CUNNINGHAM P., BONZANO A., Knowledge engineering issues in developping a case-based reasoning application , *knowledge-Based Systems*, vol. 12, 1999, p. 371-379, Elsevier.

[DEM 99] DEMEYER S., DUCASSE S., Metrics, Do they really help ? , *Langages et modèles à objets (LMO'99)*, 1999.

[FAY 02] FAYAD M., Accomplishing Software Stability , *Communications of the ACM*, vol. 45, 2002.

[FEN 00] FENTON N. E., NEIL M., Software metrics : roadmap, in The Future of Software Engineering , *To appear in IEEE Transactions on Software engineering*, 2000.

[GEN 00] GENERO M., JIMENEZ L., PIATTINI M., Measuring the quality of entity relationship diagrams , *In Proc. of 19th International Conference on Conceptual Modeling*, 2000.

[GRO 02] GROSSER D., Construction itérative de bases de connaissances descriptives et classificatoires avec la plate-forme à objets IKBS : application à la systématique des coraux des Mascareignes , Thèse de doctorat, Université de la Réunion, Janvier 2002.

[HAK 93] HAKKARAINEN J., LAAMANEN P., RASK R., Neural Networks in Specification Level Software Size Estimation , *26th Hawaii Int. Conf. System Sciences*, 1993, p. 626-634.

[KAR 92] KARUNANITHI N., WHITLEY D., MALAIYA Y., Prediction of Software Reliability Using Connectionist Models , *IEEE Trans. Soft. Eng.*, vol. 18, 1992, p. 563-574.

[MAO 98] MAO Y., SAHRAOUI H. A., LOUNIS H., Reusability Hypothesis Verification Using Machine Learning Techniques : A Case Study , *IEEE Automated Software Engineering Conference*, 1998.

[MAR 97] MARTIN R., Stability , *C++ Report*, vol. 9, n° 2, 1997.

[PAR 94] PARNAS D., Software aging , *In Proc. 16th Int. Conference on Software Engineering (ICSE'94)*, 1994.

[PIG 97] PIGOSKI T. M., *Practical Software Maintenance*, Wiley Computer Publishing, 1997.

[PRE 97] PRESSMAN R. S., *Software Engineering, A Practical Approach*, fourth edition, McGraw-Hill, 1997.

[SAH 01] SAHRAOUI H., BOUKADOUM M., LOUNIS H., Building Quality Estimation models with Fuzzy Threshold Values , *L'objet*, vol. 17, n° 4, 2001.

[SHA 93] SHAO J., Linear model selection by cross-validation , *Journal of the American Statistical Association*, vol. 1, n° 88, 1993, p. 486-494.

[WEI 91] WEISS S., INDURKHYA N., Decision tree pruning : Biased or optimal , *12 int. conf. on Artificial Intelligence*, AAAI Press and MIT Press, 1991, p. 626-632.

[WIL 97] WILSON D., MARTINEZ T., Improved Heterogeneous Distance Functions , *Journal of Artificial Intelligence Research*, vol. 6, 1997.

[WIT 94] WITTIG G., FINNIE G., Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort , *Australian Journal of Information Systems*, 1994.

[WIT 95] WITTIG G., Estimating Software Development Effort with Connectionist Models , rapport, 1995, Monash University.

[YOU 61] YOUDEN W. J., How to evaluate accuracy , *Materials Research and Standarts, ASTM*, 1961.