Analyse de l'impact du changement : approche et étude de cas

M.K Abdi[*], H. Lounis[**]

**: Département d'Informatique, Université Es-Sénia d'Oran*
BP 1524, Oran, El M'naouer, Algérie
abdi.mustapha@univ-oran.dz

***: Département d'Informatique, Université du Québec à Montréal*
Case postale 8888, succursale Centre-ville, Montréal QC H3C 3P, Canada
lounis.hakim@uqam.ca

Résumé : Nous proposons dans cet article une approche et une démarche pour analyser et prédire les impacts des changements dans les systèmes à objets. La démarche que nous suivons consiste dans un premier temps, à choisir un modèle d'impact existant, pour ensuite l'adapter à notre contexte de travail. Une technique de calcul d'impact basée sur un méta-modèle est développée. Des données récoltées sur des systèmes réels sont envisagées pour étudier empiriquement des hypothèses de causalité entre d'une part, des attributs internes de logiciels, et d'autre part, l'impact de changement. Afin d'évaluer notre approche, une étude empirique a été menée sur un système réel dans laquelle une hypothèse de corrélation entre le couplage et l'impact de changement a été avancée. Un changement concret a été porté sur le système et des métriques de couplage ont été extraites de ce système. L'hypothèse a été étudiée par le biais de 3 techniques différentes d'apprentissage automatique. Les résultats ont montré que l'impact de changement peut être dépendant d'un type de couplage spécifique, e.g. celui d'importation.
Mots clés: Systèmes à objets, impact du changement, analyse, prédiction, métriques, apprentissage automatique.

1 Introduction

La modification des systèmes est une tâche à la fois difficile et porteuse de conséquence sur la suite de l'évolution de ces systèmes. Les effets des changements subis par le système doivent donc être pris en considération. La motivation de notre travail est d'améliorer la maintenance des systèmes orientés objet, et d'intervenir plus précisément dans la tâche de l'analyse de l'impact de changement. Nous visons principalement la réduction de l'effort ainsi que le coût de la maintenance. La réduction de cet effort peut être accomplie par la réduction du temps entre la proposition du changement, son implantation et sa réalisation, tout en assurant la qualité du système. L'effort peut être aussi réduit si on peut prédire le comportement du système face à d'éventuels changements. Notre travail se situe beaucoup plus dans cet axe de recherche. Plus l'analyse et la prédiction d'impact de changement est systématique, plus la réduction d'effort est à notre avis optimale. Ainsi les bonnes décisions peuvent être prises avant d'initier les changements. En identifiant l'impact potentiel d'une modification, on réduit le risque de s'embarquer dans des changements coûteux et imprévisibles. Plus un changement affecte de classes, plus le coût de sa réalisation est élevé. L'analyse des impacts de changement permettra ainsi d'évaluer le coût d'un changement et de faire un compromis entre les différents changements suggérés.

La section 2 résume l'état de l'art des différents travaux faits autour de l'analyse d'impact de changement. Notre proposition fait l'objet de la troisième section. Nous expliquons les points fondamentaux sur lesquels se basent notre approche ainsi que les étapes générales de la démarche adoptée. Ensuite, nous présentons le modèle d'impact de changement choisi pour mener nos travaux, suivi de son adaptation à Java. La technique de calcul d'expressions d'impact de changement basée sur une approche par méta-modèle (PTIDEJ) [GUÉ 03] finira cette section. La section 4 est réservée à l'étude empirique et à la discussion des résultats trouvés. Les perspectives de notre travail sont discutées en conclusion.

2 Travaux connexes

Plusieurs études ont été menées pour valider les métriques et les relier à certaines propriétés de la maintenabilité. Li et Henry [LI 93] ont pris cinq métriques de Chidamber & Kemerer [CHI 94] et ont ajouté trois métriques propres à eux pour montrer qu'il existe une forte relation entre ces métriques et l'effort de maintenance (exprimé par le nombre de lignes-codes changées). Lounis & al [LOU 97] ont proposé une suite de 24 métriques de code pour générer des modèles prédictifs liés à la propension d'erreurs (fault proneness) dans un système orienté objet. Enfin, dans [BRI 01], les auteurs ont étudié aussi les relations entre la majorité des métriques de couplage, de cohésion et d'héritage et la propension d'erreurs des classes de systèmes orientés objets.

Moins de travaux ont été menés concernant l'impact de changement. Dans [ANT 99], les auteurs ont prédit la taille des systèmes orientés objets évoluant dans le temps en se basant sur l'analyse des classes impactées par une demande de changement. Kung & al [KUN 95] intéressés par les tests de régression, ont développé un modèle d'impact de changements basé sur les trois liens : héritage, association et agrégation. Li et Offutt [LEE 96] et [LEE 98] pour examiner les effets d'encapsulation, d'héritage et de polymorphisme sur l'impact de changements, ont proposé des algorithmes pour calculer l'impact complet de changements faits dans une classe donnée. Dans [CAN 01], l'analyse d'impact a été faite avec l'objectif de réduire les coûts et la durée des tests de régression. L'analyse a été faite à partir d'un graphe de dépendance. Briand & al dans [BRI 99], ont essayé de voir si les mesures de couplage capturant toutes sortes de collaborations entre classes, peuvent aider à faire l'analyse d'impact de changement. La stratégie adoptée dans cette étude est différente des autres stratégies dans la mesure où elle est purement empirique. Dans [CHA 98] et [KAB 02], un modèle de changements et d'impact de changements, a été défini au niveau conceptuel pour étudier la changeabilité des systèmes à objets. Selon une perspective différente, Sahraoui & al. ont étudié dans [SAH 00], l'impact du refactoring sur la structure et donc sur les métriques structurelles. Cette étude a permis de déterminer quels sont les refactorings qui peuvent améliorer ou détériorer certaines propriétés structurelles.

En résumé, les études faites dans [BRI 99] et [WIL 99] sont des exemples d'approches purement empiriques. Les travaux [KUN 95], [LEE 96], [CAN 01] et [LEE 98] relèvent de la catégorie d'approches basées principalement sur des modèles qui sont des graphes de dépendance et éventuellement enrichis de certains formalismes. Les études [CHA 98] et [KAB 02] proposent un modèle différent. Nous en parlerons par la suite. Nous avons remarqué à travers cette synthèse qu'il y a plus de travaux à base de graphe de dépendance que de travaux à base d'autres modèles ou purement empiriques, ce qui explique qu'il vaudrait mieux à notre avis explorer d'autres voies de recherche. D'autre part, le plus souvent, l'impact n'est pas calculé d'une façon systématique et enfin, nous soulignons le fait qu'il y a eu beaucoup plus d'expérimentations sur des petits systèmes que sur des systèmes réels de taille industrielle, ce qui par conséquent, empêche la généralisation des résultats trouvés (règles, relations, lois, etc.). Dans la section suivante, nous présentons notre proposition en commençant par expliquer l'approche globale puis la démarche adoptée.

3 Proposition

3.1 Approche globale

Nous avons décidé dés le début que notre approche ne sera pas purement empirique pour la simple raison que nous visons des résultats (règles, relations de cause à effet, etc.) plus au moins généraux, ou à la limite qui peuvent s'appliquer à des domaines d'application larges. Cela ne réduit pas l'importance et la nécessité de la voie empirique dans notre approche; elle est à la fois analytique et expérimentale. L'étude des changements et de l'analyse de leurs impacts doivent se faire, à notre avis, d'abord à un niveau plus haut d'abstraction. Les résultats trouvés à ce niveau doivent être nécessairement testés et vérifiés par la suite par le biais d'études empiriques. Une remise en cause des modèles utilisés au niveau abstrait (conceptuel) est tout à fait possible dans le cas où les études empiriques n'aboutissent pas aux résultats proposés par les modèles, sinon une explication doit être donnée aux exceptions des résultats trouvés. Suite à notre revue de la littérature du domaine, nous avons remarqué qu'il y a très peu de travaux qui proposent un modèle d'impact de changements plus au moins complet, dans le sens où le modèle tient compte des principaux liens qu'on peut trouver dans une conception orientée objet (à savoir l'association, l'agrégation, l'invocation et l'héritage) [ANT 99].

Dans notre cas, nous avons repris le modèle d'impact défini dans le projet SPOOL[1] [SCH 01], [CHA 99] et [KAB 02] vu que ce modèle est l'un des plus généraux et surtout du fait qu'il permet de calculer l'impact d'une façon systématique, ce qui est à notre avis, un facteur important dans la réduction de l'effort ainsi que celle du coût de la maintenance. Le projet SPOOL avait comme objectif principal la compréhension des propriétés de conception des systèmes industriels et de leurs influences sur la maintenance et l'évolution de ces derniers.

Dans notre travail, nous utilisons ce modèle pour mener nos expérimentations. Ces dernières doivent être, à notre avis, orientées dans la mesure où elles doivent vérifier des hypothèses énoncées auparavant. Ces hypothèses manquent évidemment de preuve. Par analogie à d'autres travaux faits dans le domaine [LOU 97] et [BRI 01], les hypothèses sont en général des relations entre certaines caractéristiques de conception (ou propriétés architecturales : cohésion, couplage, etc.) d'un système et l'impact de changement dans notre cas. Ces caractéristiques de conception sont mesurées par des métriques. Le choix de ces métriques fait partie de l'orientation des études empiriques.

La vérification de ces hypothèses peut être faite de différentes manières, comme par exemple, par le biais de techniques à base de modèles statistiques [KAB 02]. Dans notre travail, nous allons opter pour des techniques de l'intelligence artificielle, plus précisément celles de l'apprentissage automatique (machine- learning) pour deux raisons principales. D'une part, ces techniques n'ont pas encore été utilisées dans des travaux antérieurs traitant de l'analyse d'impact de changement. D'autre part, le résultat de l'utilisation de ces techniques est un ensemble de connaissances représentées selon un formalisme, qui peut être exploité par un système de décision basé sur les connaissances. Enfin, nous signalons que comme nous visons les systèmes logiciels codés en Java, une opération d'adaptation du modèle utilisé (modèle défini au niveau conceptuel) à ce langage s'avère nécessaire afin de pouvoir calculer l'impact de tout changement atomique possible en Java. La section suivante résume les points essentiels de notre approche globale et détaille la démarche adoptée.

3.2 Démarche

Les principales actions à entreprendre dans le cadre de notre approche sont les suivantes :
1. Choisir un modèle défini au niveau conceptuel.
2. Adapter ce modèle à Java.
3. Prendre des systèmes réels et appliquer concrètement un ou des changements (au niveau du code).
4. Définir l'expression ou les expressions d'impact de changement.
5. Formuler des hypothèses de causalité reliant des caractéristiques internes des applications et l'impact de changement.
6. Dériver les métriques à partir des hypothèses formulées.
7. Vérifier les hypothèses par des techniques d'apprentissage automatique.

Notons que le retour depuis l'étape 7 aux trois premières étapes est tout à fait possible dans un but de remise en cause.

Afin de concrétiser cette démarche, nous avons besoin de systèmes réels de taille industrielle pour faire nos expérimentations. La diversité des domaines d'applications est souhaitable dans notre travail. Nous avons besoin aussi d'outils (efficaces) qui permettent d'analyser le code du système sous test. La programmation de l'expression (ou des expressions) calculant l'impact de changement déduit par le modèle au niveau conceptuel dépend d'une part du (des) changement (s) considéré (s), et d'autre part, de l'outil d'analyse utilisé. Le calcul des métriques choisies ou leur programmation en cas de nouvelles métriques est une tâche qui peut être réalisée dans le cadre de l'outil utilisé, et sera ainsi une partie intégrante de l'outil, comme elle peut lui être totalement indépendante. Enfin, le choix de techniques d'apprentissage automatique dans un but de vérification d'hypothèses dépend de leurs performances.

3.3 Modèle d'Impact de Changement

[1] SPOOL: "Spreading Desirable Properties into the design of Object-Oriented Large-scale software systems". Ce projet SPOOL a été organisé par CSER (Consortium for Software Engineering Research), et subventionné par BELL Canada, NSERC (Natural Sciences and Research Council of Canada) et NRC (National Research Council Canada).

3.3.1 Objectifs

Nous nous concentrons sur comment les divers liens entre des classes influencent en fait l'impact de changement. Cela permet de s'assurer que le système s'exécutera toujours correctement après que le changement soit mis en œuvre. Notre intérêt est concentré sur comment le système réagit à un changement (en général). Notons qu'un système absorbe facilement un changement si le nombre de composants impactés est petit.

3.3.2 Modèle Conceptuel

Un système est vu comme un ensemble de classes connectées par différents liens. Une classe est définie comme un groupe de méthodes qui servent comme interface publique ou pour des opérations internes, et une section de variables qui définissent l'état des instances de la classe.

3.3.2.1 Changements

Nous définissons un changement à un système comme un changement qui peut s'appliquer à un composant. Un composant se réfère à une classe, une méthode, ou bien une variable. Comme exemples de changement, on peut avoir la suppression d'une variable, le changement de la portée d'une méthode, de "public" à "protected", ou le déplacement du lien entre une classe et son parent.
La table 1 consigne les principaux changements aux systèmes orientés objets, au niveau conception. Ils sont définis puis classifiés selon le composant qu'ils affectent et un total de 13 changements est identifié.

Composant	***Description du Changement***
Variable	Changement de type de variable
	Changement de portée de variable
	Ajout de variable
	Suppression de variable
Méthode	Changement de type de retour de méthode
	Changement d'implémentation de méthode
	Changement de signature de méthode
	Changement de portée de méthode
	Ajout de méthode
	Suppression de méthode
Classe	Changement de structure d'héritage de classe
	Ajout de classe
	Suppression de classe

Table 1. *Principaux changements au niveau conceptuel*

3.3.2.2 Liens

Une fois qu'un composant donné est soumis à un changement, une partie spécifique peut être affectée, dans le cas où elle est liée au composant changé via un lien. Ces liens sont parmi les quatre types suivants :
S (association) : une classe fait référence aux variables d'une autre classe, **G** (agrégation) : la définition d'une classe implique des objets d'une autre classe, **H** (héritage) : une classe hérite les particularités définies dans une autre classe parente),
I (invocation) : les méthodes d'une classe invoquent des méthodes définies dans une autre classe.

Nous considérons aussi une notation spéciale généralement utilisée dans l'algèbre booléenne : L'absence d'un opérateur entre 2 liens signifie une ***intersection***. L'opérateur "**+**" signifie une ***union***. L'opérateur "**~**" avant un lien signifie la ***négation***, *c'est-à-dire l'*ensemble des classes non associées par ce lien spécial, par exemple, **~G** signifie l'ensemble des classes qui ne sont pas liées à la classe indiquée par le lien d'agrégation. Les liens sont indépendants les uns des autres et nous pouvons trouver n'importe quel nombre et type de liens entre deux classes. Un changement d'une classe peut aussi avoir un impact dans la même classe. Le pseudo-lien **L** (local) est introduit pour exprimer ceci.

3.3.2.3 Impact

Nous appelons impact d'un changement l'ensemble des classes qui exigent une correction suite à ce changement. Il dépend de deux facteurs : l'un est le type de changement. Par exemple, un changement de type de variable a un impact sur toutes les classes faisant référence à cette variable, tandis que l'ajout d'une variable n'a aucun impact sur ces classes. Étant donné un type de changement, l'autre facteur est la nature des liens impliqués. Si, par exemple, la portée d'une méthode est changée de "public" à "protected", les classes qui invoquent la méthode seront impactées, à l'exception des classes dérivées. Nous notons que plus qu'un type de lien entre la classe changée et une classe impactée peut être impliqué dans le calcul de l'impact. Ainsi, pour un changement donné ch_i dans la classe cl_j, l'ensemble des classes impactées est exprimé par une expression booléenne dans laquelle les variables représentent les liens. Par exemple, la formule d'impact pour un changement hypothétique peut être donnée par :

$$\text{Impact } (cl_j, ch_i) = \mathbf{S\sim H+G}$$

Cette expression signifie que les classes qui sont en association (S) avec la classe changée cl_j et non dérivées (~H) de cette classe, ou les classes qui sont en agrégation (G) avec cl_j : sont impactées.
Dans notre travail, nous nous intéressons seulement aux changements qui ont un impact syntaxique. Un changement donné est caractérisé par une transformation du code quelque part dans le système. Si le système est recompilé avec succès, alors il n'y a aucun impact. Sinon, nous sommes face à un impact, c'est-à-dire, des modifications du code qui doivent être faites ailleurs dans le système pour obtenir un code syntaxiquement correct qui se recompilera.

3.4 Adaptation du modèle à Java

Nous signalons que ce modèle défini au niveau conceptuel a été déjà adapté à C++. Cela représentait une contrainte du partenaire industriel du projet SPOOL [CHA 98], [CHA 99] et [SCH 01], projet dans lequel a été défini ce modèle d'impact. Comme dans notre travail, nous visons les systèmes logiciels codés en Java, une opération d'adaptation de ce modèle à ce langage s'avère nécessaire. Nous avons examiné l'adaptation déjà faite dans [CHA 98] et [KAB 02]. Nous avons remarqué qu'il y a certains changements communs aux deux langages. Nous citons comme exemples, le changement de type de variable, changement de signature de méthode, changement de la structure d'héritage d'une classe, etc. Par contre, il y a d'autres changements qui sont propres au langage C++. Ces derniers concernent principalement les concepts de "virtual" (méthode virtuelle ou classe virtuelle) et "friendship" (classe amie). Le concept de "virtual" est introduit en C++ pour gérer les appels dynamiques. Le rôle joué par ce concept en C++ est assuré en Java par le biais de sa machine virtuelle. Le concept d'amitié (friendship) n'existe pas en Java. En résumé, les changements propres à C++ viennent des concepts qui lui sont propres. Nous tenons à souligner à ce niveau, que notre intérêt dans cette étude est de voir les changements qui peuvent être appliqués en Java. Donc, mis à part les changements touchant à ces deux concepts, le reste des changements raffinés en C++ sont tout à fait possible dans le langage Java. La liste finale contient un total de 52 changements, comprenant 12 changements pour la variable, 25 pour la méthode et 15 pour la classe. Dans la section suivante, nous parlons de l'outil utilisé ainsi que de son extension pour répondre à notre objectif de calcul d'impact.

3.5 Outil utilisé : PTIDEJ

Pour nos expérimentations, nous avons opté pour PTIDEJ[2] [GUÉ 03] afin d'analyser le code des programmes (systèmes) considérés. Guéhéneuc propose et décrit dans sa thèse des modèles et des algorithmes pour garantir la traçabilité des motifs de conception[3] entre les phases d'implantation et de rétroconception des programmes. Il fait cela par l'identification semi-automatique de micro-architectures similaires à ces motifs dans le code source des programmes. Ces modèles et ces algorithmes forment un cadre pour la traçabilité des motifs de conception. La suite d'outils PTIDEJ est une implantation en Java de ces modèles et algorithmes. Elle est intégrée à l'environnement de développement (*EDI*) Eclipse [OTI 01]. Elle permet la modélisation des programmes Java, l'identification et la traçabilité des relations

[2] Ptidej : Pattern Trace Identification, Detection, and Enhancement in Java.

[3] Un motif de conception est la solution d'un patron de conception.

interclasses (association, agrégation et composition) et des motifs de conception entre les phases d'implantation et de rétroconception.

Elle inclut le métamodèle PADL[4], dérivé du métamodèle PDL[5] défini dans une autre thèse [ALB 03], pour modéliser les relations interclasses, les motifs de conception et les programmes Java. Le métamodèle PADL dispose d'un ensemble de constituants nécessaires à la description des modèles d'un programme aux niveaux implémentation, idiomatique[6] et conception. Le métamodèle PADL est utilisé aussi pour modéliser un motif de conception au niveau idiomatique. Cette modélisation consiste à décrire ses participants et leurs relations avec les constituants du métamodèle PADL, et fournit par la suite un modèle abstrait du motif. L'implantation du patron de conception Visiteur intègre l'interface Walker qui joue le rôle de visiteur. Cela permet de parcourir tous les constituants d'un modèle de programme exprimé avec PADL. Ce mécanisme de visiteur est utilisé par exemple pour calculer des métriques sur les modèles de programmes, ou générer des informations comme la liste globale des entités du modèle. Pour plus de détails sur l'outil PTIDEJ et ses métamodèles, nous orientons le lecteur vers [GUÉ 03] et [ALB 03].

En ce qui nous concerne, notre intervention réside à ce niveau du modèle PADL pour l'étendre afin qu'il puisse répondre à nos besoins de calcul d'expressions d'impact de changements. Nous avons implémenté, pour le moment, les classes permettant de déduire les impacts de changements dont l'expression résultat fait appel aux liens d'agrégation, d'association et d'héritage. Le lien d'invocation (de méthodes) n'a pas été pris en considération dans notre travail pour la simple raison que la version du méta-modèle utilisée ne l'offrait pas. Nous signalons que nous prenons en charge ce lien dans une perspective à court terme.

4. Étude empirique

4.1. Objectif

Comme les liens inter-classes sont censés être plus responsables de la propagation de la modification que les liens intra-classe, nous allons nous concentrer sur la propriété de couplage, et voir s'il y a des relations de cause à effet entre cette propriété architecturale du système et l'impact de changement. Nous proposons l'hypothèse suivante : *"Le couplage influence l'impact de changement dans un système objets"* Nous avons cité dans la section 2 plusieurs travaux autour de cette propriété architecturale mais l'objectif dans cette expérimentation est de voir quels types de couplage influence le plus l'impact de changement.

4.2 Système considéré

Nous avons choisi pour cette étude empirique, un système disponible à notre niveau. Il s'agit de BOAP (Boîte à Outils pour l'Analyse de Programmes) développé au Centre de Recherche Informatique de Montréal (CRIM) [ELH 02]. C'est un ensemble d'outils logiciels intégrés, qui permet à un expert d'évaluer rapidement le niveau de qualité d'un logiciel (faiblesses conceptuelles ou structurelles, instructions trop complexes, etc.). Le système BOAP (version 1.1.0) que nous avons considéré contient en tout 394 classes.

4.3. Changements considérés et métriques sélectionnées

Nous avons choisi comme changement : le changement de type de variable. A titre d'exemple, nous avons déterminé par le biais de la technique de calcul élaborée une classe qui présente un nombre important de liens d'associations avec les autres classes pour avoir un impact assez considérable sur le reste du système selon le changement envisagé. Ensuite, nous avons sélectionné une variable puis porté notre changement. Nous obtenons:

Classe considérée : dbClass (du package : DBLMR)

Variable choisie : sizeInBytes

[4] PADL : Pattern and Abstract-level Description Language.

[5] PDL : Pattern Description Language.

[6] Niveau idiomatique : niveau d'abstraction défini entre les niveaux implémentation et conception.

Changement : de type "**long**" au type "**integer**"
L'expression d'impact de ce changement est : S + L
Ce qui signifie qu'il y a impact de changement localement d'abord (dans la classe changée elle même) et aussi dans toutes les classes du système qui sont en association avec la classe changée "dbClass".
La technique de calcul d'impact de changement nous retourne un résultat de : 42 classes. Il y a donc 42 classes impactées suite à ce changement.
Nous avons procédé de la même façon pour le reste des classes du système. Notons qu'il peut s'agir d'autres changements de types de variables (pas nécessairement de "**long**" à "**integer**"), comme il est tout à fait possible qu'un changement ne crée aucun impact (impact nul).
Ensuite, nous avons extrait de notre système de test BOAP un ensemble de métriques reliées toutes à la propriété de couplage. Elles sont présentées dans la table 2. Nous les avons calculées par le biais de l'outil développé dans [CHE 04].

Métriques	Définition
RFC	Response For a Class : nombre de méthodes invoquées en réponse à un message.
MPC	Message Passing Coupling : nombre de messages envoyés par une classe en direction des autres classes du système.
CBOU	CBO Using : se réfère aux classes utilisées par la classe cible.
CBOIUB	CBO Is Used By : se réfère aux classes utilisant la classe cible.
CBO	Coupling Between Object : nombre de classes avec lesquelles une classe est couplée
CBONA	CBO No Ancestors : CBO sans considérer les classes ancêtres.
AMMIC	Ancestors Method–Method Import Coupling : nombre de classes parentes avec lesquelles une classe a une interaction de type méthode-méthode et un couplage de type IC.
OMMIC	Others Method–Method Import Coupling : nombre de classes (autres que les super-classes et les sous-classes) avec lesquelles une classe a une interaction de type méthode-méthode et un couplage de type IC.
DMMEC	Descendants Method – Method Export Coupling : nombre de sous-classes avec lesquelles une classe a une interaction de type méthode-méthode et un couplage de type EC.
OMMEC	Others Method – Method Export Coupling : nombre de classes (autres que les superclasses et les sous-classes) avec lesquelles une classe a une interaction de type méthode-méthode et un couplage de type EC.

Table 2. *Métriques sélectionnées*

4.4. Étude de l'hypothèse

Comme déjà signalé, nous abordons cette étude par le biais de techniques d'apprentissage automatique. Nous avons fait appel à l'environnement Weka (Waikato Environment for Knowledge Analysis) [WIT 00] pour atteindre cet objectif. Weka est un ensemble d'outils permettant de manipuler et d'analyser des fichiers de données, implémentant la plupart des algorithmes d'apprentissage automatique, dont les arbres de décision et les réseaux de neurones. Il est écrit en java, est "open source" et disponible sur le web[7]. Nous avons voulu lors de cette expérimentation, utiliser plusieurs algorithmes d'apprentissage (J48, PART et NBTree) afin de trouver diverses relations de cause à effets entre les métriques de couplage et l'impact de changement. Dans ce travail, le choix de ces algorithmes a été basé sur trois critères, à savoir la facilité de l'interprétation des modèles trouvés, la complémentarité et la précision des résultats. Rappelons que notre système contient 394 classes, que l'impact dans notre cas est quantifié par le nombre de classes impactées, et que suite à un changement (changement de type de variable) toutes les classes où il y a eu propagation de modification, ont été considérées, même s'il ne s'agit que d'impact local (ou encore impact ≥ 1). Nos données d'apprentissage regroupent 11 variables (10 variables indépendantes + la variable dépendante). Les variables indépendantes représentent les métriques de couplage que nous avons extrait du système testé. La variable dépendante représente l'impact de changement. Toutes les variables indépendantes sont numériques. Par contre, la variable dépendante est nominale. Initialement, celle-ci était numérique parce que nous l'avons calculé par le biais de notre technique de calcul, et elle est le résultat de l'expression d'impact S+L (voir section 4.3). Il était

[7] www.cs.waikato.ac.nz/ml/weka

nécessaire de la transformer en variable nominale pour pouvoir utiliser efficacement les 3 algorithmes, en particulier J48. Pour cela, nous avons divisé l'ensemble des valeurs d'impact en 5 tranches, chacune correspondante à une valeur nominale d'impact, variant de "très-faible" à la valeur " très-fort".

4.5 Résultats et discussion

J48

J48 est une implémentation de l'algorithme bien connu C4.5 [QUI 93]. C'est un algorithme d'apprentissage supervisé qui induit un modèle de classification sous la forme d'un arbre de décision ou de règles, et ce, à partir d'un ensemble d'exemples. L'étape clé de ce type d'algorithmes est le choix du "meilleur" attribut à tester afin d'obtenir des arbres de décision compacts et possédant la meilleure capacité prédictive possible. Des heuristiques basées sur la notion d'entropie ont montré leur efficacité pour réaliser ce type de choix. Lors de l'exécution de cet algorithme sur notre ensemble de données, nous avons choisi le mode de test validation croisée (cross-validation). C'est une technique dans laquelle l'ensemble des données disponibles est divisé en N blocs. Elle consiste à créer un modèle (apprendre une hypothèse) sur N-1 blocs, puis à tester ce modèle sur le bloc restant. L'algorithme refait la même chose pour chacun des N blocs de données; ainsi l'opération entière est faite N fois. Le taux de succès obtenu (73,85%) est assez intéressant, c'est-à-dire que sur 394 instances, 291 ont été correctement classifiées. Par contre, nous trouvons que l'arbre de décision généré est trop grand (taille : nombre de nœuds=67). Par conséquent, il est difficile de tirer des règles de causalité depuis cet arbre.

Dans un objectif d'avoir des résultats plus clairs, notamment un arbre de décision plus compact, nous avons opté pour un prétraitement des données. Nous avons ainsi retenu un ensemble réduit d'attributs (ou variables indépendantes), les plus pertinents, au lieu de considérer l'ensemble de tous les attributs. Pour cela, nous avons sélectionné cet ensemble réduit d'attributs par le biais de l'algorithme "CfsSubsetEval" dans l'environnement Weka toujours, avec la méthode de recherche "BestFirst", paramétrée en recherche descendante. Il s'agit d'un algorithme simple de filtrage qui range des sous-ensembles d'attributs selon une corrélation basée sur une fonction heuristique d'évaluation. Les attributs non pertinents devraient être ignorés parce qu'ils auront une faible corrélation avec la variable à prédire. Les attributs qui ont été sélectionnés sont : MPC, CBOU, CBONA, AMMIC, et OMMIC (voir table 2). Nous avons re-exécuté l'algorithme J48 sur notre ensemble de données ainsi réduit. Le taux de succès obtenu (73,30 %) est très proche du précèdent. Par contre, l'arbre de décision obtenu est bien plus compact (nombre de nœuds = 31). Il contient 16 feuilles. Chaque chemin de la racine à une feuille donnée est une règle de causalité. Nous avons donc un ensemble de 16 règles. La figure 1 présente certaines règles choisies de cet ensemble.

Règle 1 : *MPC ≤ 21*
OMMIC ≤ 4
AMMIC = 0
→ impact: faible (119.0/51.0)

Règle 2 : *MPC ≤ 21*
OMMIC ≤ 4
AMMIC > 3
→ impact: faible (32.0)

Règle 11 : *MPC ≤ 21*
OMMIC > 4
CBOU ≤ 7
→ impact: faible (68.0/12.0)

Règle 12 : *MPC ≤ 21*
OMMIC > 4
CBOU > 7
→ impact: moyen (9.0/1.0)

Règle 15 : *MPC > 36*
CBOU > *14*
AMMIC ≤ 5
→ impact: moyen (4.0/1.0)

Règle 16 : *MPC > 36*
CBOU > *14*
AMMIC > 5
→ impact: très-fort (2.0)

Figure 1. *Règles de causalité (J48)*

La première remarque à faire sur cet ensemble de règles est qu'il y a 14 règles sur 16, où les métriques de couplage d'importation sont impliquées. Cela montre bien l'influence de cette propriété particulière de couplage sur l'impact de changement. En observant bien ce sous ensemble de 14 règles, on peut encore distinguer 3 sous-ensembles particuliers. Le premier sous-ensemble est formé des 10 premières règles, où figurent dans la partie gauche les deux métriques de couplage d'importation OMMIC et AMMIC. Le deuxième est formé des règles 11 et 12, où figurent seulement la métrique OMMIC. Le troisième est formé des règles 15 et 16, où figurent seulement la métrique AMMIC. Dans le premier sous ensemble, on remarque que dans la plupart des cas, l'impact est faible ou très-faible pour les classes qui présentent un faible couplage d'importation Méthode-Méthode-Autres (OMMIC ≤ 4, la moyenne est 9.26) et un faible/moyen couplage d'importation Méthode-Méthode-Ancêtres (AMMIC entre 0 et 3, la moyenne est

2.15). Dans le second sous-ensemble, la métrique OMMIC n'est pas déterminante mais elle représente un élément important à considérer dans la prédiction de l'impact. Ce dernier est faible ou moyen selon que le nombre de classes utilisées par la classe cible est moyen ou grand (CBOU est ≤ 7 ou > 7, la moyenne est 3.57). Cela est valable pour les classes qui disposent d'un nombre d'invocations statiques de méthodes pas très grand (MPC ≤ 21, la moyenne est 11.34) et d'un couplage d'importation Méthode-Méthode-Autres pas trop petit (OMMIC > 4, la moyenne est 9.26). Enfin, dans le troisième sous-ensemble, les règles expriment que pour les classes dont le nombre d'invocations statiques de méthodes est grand (MPC > 36) et dont le nombre de classes utilisées par la classe cible est grand aussi (CBOU>14), le couplage d'importation Méthode-Méthode-Ancêtres est déterminant dans le sens où l'impact pour ces classes sera moyen ou très-fort selon la valeur de cette propriété architecturale, quelle soit moyenne (autour de 2.5 et ≤ 5) ou grande (>5). Cela représente un résultat important de cette expérimentation.

PART

PART [FRA 98] permet d'inférer des règles par la génération itérative d'arbres de décision partiels en combinant deux paradigmes majeurs : les arbres de décision et la technique d'apprentissage des règles "diviser pour régner". La combinaison de ces deux paradigmes ajoute de la flexibilité et de la vitesse. En effet, il est inutile de construire un arbre de décision plein pour obtenir une règle simple. Le processus peut être accéléré sensiblement sans sacrifier les avantages des deux approches. L'idée principale est de construire un arbre de décision *partiel* au lieu d'un arbre entièrement exploré. PART fournit des résultats aussi précis que ceux de l'algorithme J48. Il permet d'éviter l'optimisation globale de ce dernier et fournit un ensemble de règles compactes et exactes.

Suite à l'exécution de l'algorithme PART sur notre ensemble de données, le taux de succès obtenu (65.48 %) semble assez bon, tout en étant plus faible que celui obtenu par J48, et l'ensemble résultat est formé de 25 règles. La première constatation à faire sur cet ensemble est que sur les 25 règles, il y a 16 règles où les métriques de couplage d'importation sont impliquées. Cela confirme la remarque faite auparavant (l'influence de cette propriété particulière de couplage sur l'impact de changement). D'autre part, plusieurs règles sont similaires à certaines trouvées par J48. Nous citons à titre d'exemples les règles 3 et 4 au niveau de PART, et 2 et 11 au niveau de J48. Nous tenons à signaler que pour les deux algorithmes, les règles 15 sont identiques. Cela confirme partiellement le résultat important trouvé par J48 (voir la fin de la section précédente). La figure 2 montre quelques règles choisies parmi l'ensemble des règles générées par PART.

Règle 3 : MPC <= 13
AMMIC > 3
→Impact: faible (33.0/1.0)

Règle 4 : MPC <= 13
OMMIC > 4
CBOU <= 1
→Impact: faible (6.0)

Règle 15 : MPC > 36
CBOU > 14
AMMIC <= 5
→Impact: moyen (4.0/1.0)

Figure 2. *Règles de causalité (PART)*

NBTree (Naïve-Bayes decision-Trees)

Kohavi [KOH 96] propose NBTree comme une approche hybride combinant le classificateur Bayésien naïf et le classificateur à base d'arbre de décision. Ce classificateur hybride obtient fréquemment une très haute précision par rapport au classificateur Bayésien naïf ou au classificateur de type arbre de décision. Il utilise une structure arborescente pour diviser l'espace d'instances en sous-espaces et générer un classificateur Bayésien naïf pour chaque sous-espace. Dans un arbre de décision conventionnel, chaque feuille est marquée avec une seule classe et prédit cette classe pour les instances qui atteignent la feuille, alors qu'un arbre Bayésien naïf utilise un classificateur Bayésien naïf local pour prédire les classes de ces instances. Suite à l'exécution de l'algorithme NBTree sur notre ensemble de données, le taux de succès obtenu est assez intéressant (66.75%). L'arbre de décision généré est compact (nombre de nœuds = 17). Il contient 9 feuilles, contenant chacune un classificateur Bayésien permettant de déduire la classe à prédire avec plus de précision. Chaque chemin de la racine à une feuille donnée est une règle de causalité. Nous avons donc un ensemble de 9 règles. La figure 3 présente quelques règles de cet ensemble. Notons qu'au niveau de la conclusion de chacune des règles, on trouve le numéro du

classificateur Bayésien naïf (Naïve-Bayes), soit NB3 pour la règle 1, suivi de la classe à prédire, qui est en fait la classe qui a la probabilité la plus élevée. Cette probabilité est mentionnée entre parenthèses. Les résultats de cet algorithme affirment qu'en plus du couplage d'importation, l'impact est aussi influencé par le couplage mesuré par les métriques CBONA et CBOU vu que ces dernières figurent dans la plupart des règles trouvées. Les règles 1 et 9 (voir figure 3) montrent bien que l'impact est très-faible ou fort selon les valeurs de ces métriques qu'elles soient petites ou grandes. Enfin, les règles 2 et 3 expriment que l'impact devient de plus en plus faible si le couplage d'importation Méthode-Méthode-Ancêtres (AMMIC) augmente. Cela confirme bien un résultat déjà trouvé par J48.

Règle 1 : CBONA ≤ 3.5
CBOU ≤ 0.5
→ NB3 : impact: très-faible (0.46)

Règle 2 : CBONA ≤ 3.5
CBOU $\in$ $]0.5,1.5]$
$AMMIC \leq 0.5$
→ NB5 : impact: faible (0.54)

Règle 3 : CBONA ≤ 3.5
CBOU $\in$ $]0.5,1.5]$
$AMMIC > 0.5$
→ NB6 : impact: très-faible (0.76)

Règle 9 : CBONA > 3.5
CBOU > 36.5
→ NB16 : impact: fort (0.48)

Figure 3. *Règles de causalité (NBTree)*

5 Conclusion

Nous avons proposé dans cet article une approche et défini une démarche afin de répondre à la problématique d'analyse et de prédiction de l'impact des changements dans un système à objets. Une étude approfondie et une synthèse générale des différents travaux antérieurs traitant de ce sujet étaient indispensables. Afin de concrétiser notre approche, nous avons choisi un modèle d'impact existant et nous l'avons adapté au langage Java. Par la suite, nous avons proposé une technique de calcul d'expressions d'impact de changement en utilisant une approche par méta-modèle (PTIDEJ). Cette technique prend en charge tout changement dont l'expression d'impact fait appel aux liens d'association, d'agrégation et d'héritage. Pour vérifier notre approche, nous avons réalisé une étude empirique dans laquelle nous avons avancé une hypothèse de corrélation entre le couplage et l'impact de changement. L'expérimentation a été faite sur le système BOAP (système développé au CRIM). Il contient en tout 394 classes. Par la suite, un changement a été concrètement porté sur ce système, il s'agit du changement de type de variable. L'impact de ce changement est déduit par le modèle utilisé puis calculé par le biais de la technique proposée. Un ensemble de métriques reliées à la propriété de couplage a été extrait du système BOAP. Nous avons utilisé 3 algorithmes d'apprentissage de l'environnement Weka pour vérifier notre hypothèse.

Les taux de succès obtenus pour les 3 algorithmes semblent assez intéressants. Les résultats trouvés par J48 puis confirmés par PART, expriment que le couplage d'importation influence beaucoup plus l'impact de changement que les autres types de couplage vu que dans la plupart des cas (règles obtenues), l'impact est principalement lié à ce type de couplage. Aussi, comme autre résultat important de cette expérimentation, il s'avère que pour les classes dont le nombre d'invocations statiques de méthodes ainsi que le nombre de classes utilisées par la classe cible sont grands, le couplage d'importation (mesurée par la métrique AMMIC) décide de l'impact de changement. Ce résultat a été trouvé par J48 puis partiellement confirmé par PART. Enfin, les résultats de NBTree ont ajouté plus de précisions aux résultats déjà trouvés par J48 et PART, et ont montré qu'en plus du couplage d'importation, l'impact est aussi influencé par le couplage mesuré par les métriques CBONA et CBOU.

Comme travaux futurs, nous envisageons d'autres expérimentations sur d'autres systèmes afin de confirmer encore plus ces résultats. Nous nous intéressons aussi à d'autres mesures de couplage, ainsi qu'à d'autres types de propriétés architecturales expliquant mieux les mécanismes les plus communs aux propagations de modification (ripple-effect) à travers les systèmes à objets. Il serait intéressant, à notre avis, de comparer l'impact de changements pour des systèmes différents et trouver ainsi des résultats applicables à une large catégorie de systèmes.

Références

[ALB 03]: ALBIN-AMIOT H., "Idiomes et Patterns Java : Application à la Synthèse de Code et à la Détection". Thèse de doctorat, université de Nantes, février 2003.

[ANT 99]: ANTONIOL G., CANFORA G., LUCIA A. D., "Estimating the size of changes for evolving object OrientedSystems: a Case Study" in Proceedings of the 6th International Software Metrics Symposium, pages 250-258, Boca Raton, Florida, Nov 1999

[BRI 99]: BRIAND L. C., WÜST J., LOUNIS H., "Using Coupling Measurement for Impact Analysis in Object-Oriented Systems" in proceedings of the International Conference on Software Maintenance ICSM'99, Oxford, England, August 30 – September 3, 1999.

[BRI 01]: BRIAND L. C., WUST J., H. LOUNIS H., "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs". In Empirical Software Engineering, an International Journal, 6 (1):11-58, March 2001, Kluwer Academic Publishers.

[CHA 98]: CHAUMUN M. A., "Change Impact Analysis in Object-Oriented Systems: Conceptual Model and Application to C++". Master's thesis, Université de Montréal, Canada, November 1998.

[CAN 01]: CANTAVE R., "Abstractions via un modèle générique d'application orientée objet", Master's thesis, Université Laval, Canada, Avril 2001

[CHA 99]: CHAUMUN M. A., KABAILI H., KELLER R. K., LUSTMAN F., "A Change Impact Model for Changeability Assessment in Object-Oriented Software Systems". In Proceedings of the Third Euromicro Working Conference on Software Maintenance and Reengineering CSMR'99, pages 130-138, Amsterdam, The Netherlands, March 1999.

[CHI 94]: CHIDAMBER S. R., KEMERER C. F., "A Metrics Suite for Object Oriented Design" in IEEE Transactions on Software Engineering, Vol. 20, No. 6, pages 476-493, June 1994.

[CHE 04]: CHEÏKHI L., "Estimation de l'impact du changement dans les programmes à Objets", Master's thesis, Université de Montréal, Canada, November 2004.

[ELH 02] : EL HACHEMI A., SNOUSSI H., "BOAP 1.1.0 : Manuel d'utilisation", CRIM, Janvier 2002.

[FRA 98]: FRANK E., WITTEN I.H., "*Generating Accurate Rule Sets Without Global* Optimization" in Proceedings of the Fifteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[GUÉ 03]: GUÉHÉNEUC Y., "Un cadre pour la traçabilité des motifs de conception", Thèse dedoctorat de l'université de Nantes, École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes, juin 2003.

[KAB 02]: KABAILI H., "Changeabilité des logiciels orientés objet : propriétés architecturales et indicateurs de qualité", PhD thesis, Université de Montréal, Canada, Janvier, 2002.

[KOH 96]: KOHAVI R., "*Scaling up the accuracy of naive-Bayes classifiers: a decision tree hybrid*" in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, (1996).

[KUN 95]: KUNG D. C., GAO J., HSIA P., LIN J., TOYOSHIMA Y., "Class firewall, test order, and regression testing of object-oriented programs" in Journal of Object-Oriented Programming, Vol. 8, No. 2, pages 51-65, May 1995.

[LEE 96]: LEE M., OFFUTT A. J., "Algorithmic Analysis of the Impact of Changes to Object-Oriented Software" in ICSM96, pages 171-184, 1996.

[LEE 98]: LEE M., "Change Impact Analysis for Object-Oriented Software". PhD thesis, George Mason University, Virginia, USA, 1998

[LI 93]: LI W., HENRY S., " Object-Oriented Metrics that Predict Maintainability" in Journal of Systems and Software, Vol. 23, pages 111-122, 1993

[LOU 97]: LOUNIS H., SAHRAOUI H. A., MELO W. L., "Defining, Measuring and Using Coupling metrics in Object-Oriented Environment" in SIGPLAN OOPSLA'97 Workshop on Object-Oriented Product Metrics, 1997, Atlanta, Georgia, USA, 1997.

[OTI 01]: Object Technology International, Inc. / IBM. Eclipse platform – A universal tool platform, July 2001.

[QUI 93]: QUINLAN J.R., "C4.5: Programs for Machine Learning". Morgan Kaufmann Publishers, Sao Mateo, CA, 1993.

[SAH 00] SAHRAOUI H. A., GODIN R., MICELI T., "Can metrics help to bridge the gap between the improvement of OO design quality and its automation ?", *in Proceedings of the International Conference on Software Maintenance (ICSM'00)*, 2000

[SCH 01]: SCHAUER R, KELLER R. K., LAGUË B., KNAPEN G., ROBITAILLE S., SAINT-DENIS G., "The SPOOL Design Repository: Architecture, Schema, and Mechanisms. In Hakan Erdogmus and Oryal Tanir, ditors, Advances in Software Engineering. Topics in Evolution, Comprehension, and Evaluation. Springer-Verlag, 2001.

[WIL 99]: WILKIE F. G., KITCHENHAM B. A., "Coupling Measures and Change Ripples inC++ Application Software", published in the proceedings of EASE'99, University of Keele, UK, 1999.

[WIT 00]: WITTEN I. H., FRANK E., "Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation", © 2000 Morgan Kaufmann Publishers.