Vers un modèle de prédiction de la qualité du logiciel pour les systèmes à objets

Hakim Lounis[Φ], Houari A. Sahraoui[Φ], Walcélio L. Melo[Ψ]

ΦCentre de Recherche Informatique de Montréal
1801 McGill College Montréal (Québec) H3A 2N4 CANADA

ΨOracle do Brasil and
Universidade Católica de Brasília
SCN Qd.02 Bl. A Sala 604 Brasilia, DF, Brazil 70712-900

{hsahraou, hlounis}@crim.ca, wmelo@oracle.com

RESUME : La modularité est considérée comme un critère important de la qualité du logiciel. Un produit logiciel est dit modulaire si ses composants présentent un haut degré de cohésion et un faible degré de couplage. Dans le cadre des applications orientées objets (OO), il existe différents types de couplages entre classes. Mesurer ces types de relations peut nous permettre de mieux comprendre le lien qui existe entre le couplage des classes et les attributs de qualité. Dans cet article, notre but est de valider empiriquement l'hypothèse selon laquelle certaines formes de couplage sont préjudiciables à la qualité du logiciel, en utilisant une méthode statistique et une technique d'apprentissage symbolique.

ABSTRACT : Modularity has been considered an important software product quality criterion. A software product is considered modular if its components exhibit a high cohesion and its components are weakly coupled. A module has high cohesion if all of its elements are related strongly. In the particular context of object oriented applications, there are different types of coupling between classes. Measuring this types of coupling can helps better understanding the relationship between class coupling and software quality attributes. In this paper, our goal is to validate empirically the hypothesis that some types of coupling have an impact on class error-proneness, using a statistical method and a machine learning technique.

MOTS-CLES : mesures de couplage entre classes, modèles prédictifs de qualité, validation statistique, algorithmes d'apprentissage symbolique.
KEY WORDS : Class coupling measures, quality predictive models, statistical validation, machine learning algorithms.

1. Introduction

D'une manière générale, la modularité est considérée comme un critère important de la qualité du logiciel. Sommerville présente la modularité comme étant un facteur important pouvant influer sur les différentes caractéristiques de qualité comme l'efficacité, la flexibilité, l'interopérabilité, la facilité de maintenance et la facilité de réutilisation [SOM95]. Un produit logiciel est dit modulaire si ses composants présentent un haut degré de cohésion ainsi qu'un faible degré de couplage [CON79]. Un module possède un haut degré de cohésion si tous ces éléments sont reliés entre eux[1]. Ces éléments collaborent pour réaliser un objectif commun qui est la "fonction" du module. Le couplage lui, concerne les relations qu'a un module avec les autres modules du système. La mesure de ce couplage détermine l'interdépendance entre deux modules.

Intuitivement, un faible degré de couplage entre les modules est souhaitable. En effet, un module qui est relativement peut relié aux autres est facile à comprendre, à modifier, à tester et enfin à réutiliser. De plus, les chances de propagation des erreurs d'un module à un autre se trouvent d'autant plus diminuées que le nombre de liens entre ces deux modules est petit.

Dans le cadre des applications orientées objets, il existe différents types de couplages entre classes (voir entre autres pour certains de ces types [CHI94], [DAV96] et [PRI97]). Mesurer ces types de relations peut nous permettre de mieux comprendre le lien qui existe entre le couplage des classes et les attributs de qualité. Par exemple, comprendre le lien qui existe entre certaines formes de couplage et la propension des classes à engendrer des erreur (error-proneness). Un tel lien peut avoir une influence significative sur la réduction des coûts de maintenance.

Dans cet article, notre but est de vérifier sur un système orienté objets de taille moyenne, l'hypothèse selon laquelle certaines formes de couplage (ou mesures de ces formes) ont un impact direct sur la propension des classes à engendrer des erreurs. Pour mener à bien cette vérification, il s'agit en premier lieu de définir sous quelles formes les classes peuvent être couplées dans un système à objets. Chaque forme peut avoir des influences différentes sur la qualité. Dans un second temps, nous mesurons ces différentes formes de couplage. La dernière étape consiste à définir la relation entre le couplage et la propension des classes à engendrer des erreurs (un attribut important de qualité du logiciel). Dans cette dernière étape, nous définissons un modèle prédictif en utilisant des techniques d'apprentissage symbolique supervisé.

La deuxième partie de cet article présente quelques travaux sur l'étude de l'impact du couplage sur la qualité du logiciel. Différentes métriques associées au couplage sont présentées. Dans la troisième, nous présentons une classification multi-critères des différentes formes de couplage entre classes au niveau du code. Nous proposons aussi une manière de mesurer ces formes. La quatrième partie est

[1] Par exemple, en programmation procédurale, un module est vu comme un ensemble d'éléments qui sont des routines.

dédiée à la vérification de l'hypothèse selon laquelle certaines de ces formes de couplages ont un impact sur la propension des classes à engendrer des erreurs. Cette vérification est faite sur un système à objets de taille moyenne. Dans ce cadre, nous utiliserons deux techniques, la validation statistique et l'apprentissage symbolique avec validation croisée.

2. Métriques de couplage et qualité du logiciel

L'étude du couplage, qu'il soit pour des systèmes procéduraux ou orientés objets a donné naissance à une série de travaux que l'on peut diviser en deux catégories : ceux qui étudient le couplage d'un système au niveau de la conception (i.e., design), et ceux qui se placent au niveau du code source du système ciblé.

Ainsi, Chidamber et Kemerer [CHI94] ont proposé un ensemble de métriques de conception OO dénommé MOOSE. Parmi elles, une mesure de couplage simple notée CBO (Coupling Between Object). Elle mesure le nombre de classes couplées à une classe donnée, considérant que le couplage existe lorsque dans l'une de ses méthodes, une classe fait référence à une méthode ou à un attribut d'une autre classe. Une deuxième mesure aussi simple que la première et notée RFC (Response For Class) indique le nombre de méthodes d'une classe augmenté du nombre de méthodes des autres classes auxquelles elles font référence. Basili et al. ont montré dans une étude que cinq des six mesures définies par [CHI94] étaient pertinentes pour la prédiction de la propension des classes à avoir des erreurs. De façon similaire, Brito e Abreu et Carapu [BEA94] ont défini le cadre de mesures MOOD incluant une mesure de couplage dite facteur de couplage (CF). Une classe A est considérée couplée à une classe B, si elle lui envoie un message. Ce genre de mesures demeure cependant relativement simple et de très haut niveau.

Récemment, Briand, Devanbu et Melo [BRI97] ont défini un ensemble de métriques de couplage de systèmes OO, au niveau conception. Ils tiennent compte des différents mécanismes propres au langage C++ (classe amie, spécialisation et agrégation). 3 modalités sont considérées, (i) le type de couplage : classe-attribut, classe-méthode ou méthode-méthode, (ii) la relation entre classes couplées : hiérarchique, classe amie ou autres, et (iii) la localisation de l'impact : couplage importé d'une classe ancêtre ou dont la dite classe est amie, ou exporté vers une classe descendante ou amie. Cela mena les auteurs à proposer 24 métriques de couplage qui furent valider empiriquement en analysant la relation de cause à effet avec la probabilité de détection d'erreurs dans les classes. Le processus de validation montra que certaines de ces 24 métriques pouvaient être des indicateurs de la qualité de la conception d'un système OO.

L'approche développée par Price et Demurjian [PRI97] énonce qu'il est plus souhaitable de considérer le couplage entre hiérarchies de classes, et ce dans un souci de réutilisation. Chaque classe du système est qualifiée de général si elle peut être réutilisée, ou spécifique si elle est spécifique à l'application en cours. 8 types de couplage sont alors identifiés ainsi que les actions associés pour créer des hiérarchies de classes réutilisables.

Le couplage a aussi été abordé au niveau du code source. Ainsi en est il des travaux de Page-Jones [PAG80] où 8 différentes mesures de couplage ont été identifiées. Pour chacune d'elles, les données partagées (paramètre, variable globale, etc.) sont classifiées par leur type d'utilisation. Dans des travaux plus récents, Offutt, Harrold et Kolte [OFF93] ont étendu les 8 niveaux de couplage de Page-Jones en 12 niveaux, proposant ainsi des mesures de couplage du code plus précises. Ces mesures de couplage sont définies entre paires d'unités P et Q (une unité correspondant dans ce cas à une procédure ou à une fonction). Pour chacun des niveaux de couplage, les paramètres d'appel (ou de retour) sont classifiés selon l'utilisation qui en est faite : calcul (Usage-C), prédicat (Usage-P) ou indirecte (Usage-I). Des détails et exemples de ces trois types d'utilisation seront donnés dans la section suivante.

Nos travaux [LOU97] sont inspirés des travaux de [OFF93]. Nous avons utilisé le même cadre de mesures que celui présenté par [BRI97]. En fait, notre travail peut être considéré comme complémentaire à celui de [BRI97], où furent définis différents niveaux de couplage de systèmes OO, au niveau conception. Cependant, dans ce papier, nous nous intéressons au couplage existant dans le code source de systèmes orientés objets.

3. Couplage entre classes

Les classes sont conçues pour collaborer entre elles. Ceci rend inévitable les références externes, à travers lesquelles le code d'une classe fait référence à une partie du code d'une autre classe. Cette référence peut concerner une donnée définie dans une classe et utilisée par une autre. Elle peut également se présenter sous la forme d'un appel d'une méthode d'une classe par la méthode d'une autre classe. Avant d'aller plus loin dans cette classification, nous présentons quelques définitions en partie inspirées des travaux de Briand et al. [BRI96]

3.1 Définitions de base

Un **système** S est vu comme un couple <E, R>, où E représente les éléments de S et R une relation binaire sur E ($R \subseteq E \times E$) représentant les liens entre ces éléments. E peut être par exemple l'ensemble des routines et R les appels entre ces routines.

Dans le cas particulier d'un **système à objets**, E peut représenter l'ensemble des attributs et des méthodes des classes du système et R les liens entre ces méthodes et attributs.

Partant de là, une **classe** C de S est à son tour vue comme un couple $<E_C, R_C>$, où E_C est un sous-ensemble de E et R_C un sous-ensemble de R. Il faut noter que R_C n'est pas uniquement un sous-ensemble de $E_C \times E_C$. E_C est l'ensemble des attributs et des méthodes de C.

Le sens d'un lien entre une classe et une autre classe est important. Une classe offre des services et utilise les services offerts par les autres classes. Ainsi, on

définit pour chaque classe, deux ensembles Entree(C) et Sortie(C) de la manière suivante :

$$Entree(C) = \{\langle e_1, e_2 \rangle \in R \mid e_2 \in E_C \wedge e_1 \in E - E_C\}$$

qui est l'ensemble des liens des éléments des autres classes du système avec les éléments de C (services utilisés), et

$$Sortie(C) = \{\langle e_1, e_2 \rangle \in R \mid e_1 \in E_C \wedge e_2 \in E - E_C\}$$

qui représente l'ensemble des liens des éléments de C avec les éléments des autres classes du système (services offerts).

Dans [BRI96], Briand et al. proposent que les mesures de couplage présentent quelques propriétés. Ces propriétés concernent le couplage entre modules. Dans le contexte des classes, nous ne voyons aucun inconvénient à les reprendre, étant donné que dans beaucoup de communautés, il est d'usage de considérer une classe comme un module. La première propriété est que ces mesures ne doivent pas être négatives. Elle sont toutefois nulles en l'absence de relations entre les classes.

Une autre propriété importante est que les mesures de couplage décroissent si deux classes sont fusionnées en une seule. Dans le contexte général des modules, on peut voir cette propriété comme souhaitable pour diminuer le couplage et par conséquent les risques de propagation des erreurs. Cependant, dans le contexte spécifique des classes, on ne peut prétendre fusionner des classes en se basant uniquement sur des informations structurelles et indépendamment de la sémantique de l'application. Ceci nous permet de mieux préciser l'objectif de cette étude qui est de montrer que la combinaison de certaines formes de couplage a une influence directe sur la risque d'avoir des erreurs dans le système. Dans cet article, on ne prétend pas proposer des solutions pour remédier à telle ou telle forme de couplage.

Il est à noter également que la définition des formes de couplage présentées dans cet article n'est pas totalement neutre. En effet, nous nous sommes inspiré du langage C++. Cependant, la plupart d'entre elles restent valables pour les autres langages *à objets* typés.

3.2 Les différents types de couplage entre classes

Pour collaborer ensemble dans un système, les classes sont reliées de différentes manières (voir figure 1). Ces liens peuvent être indirects dans le cas du *partage de données ou de types de données* (une donnée ou type de données défini dans une classe et utilisé dans une autre classe). Ils peuvent être considérer comme directs. C'est notamment le cas des *appels de méthodes* (une méthode d'une classe appelle une méthode d'une autre classe).

Une façon complémentaire de distinguer le couplage entre classes consiste à considérer les trois critères suivants :

- La complexité de l'information partagée (simple, complexe, etc.)
- Le mode de transmission des paramètres dans le cas des appels de méthodes (par valeur ou par référence)
- L'usage qui est fait de l'information partagée.

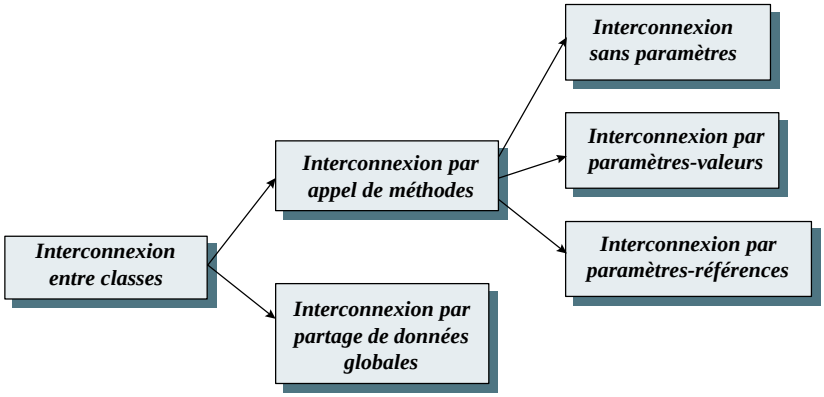

Figure 1. *Différents types de liens entre classes*

Pour le premier critère, on parle de type simple quand il s'agit d'un type de base du langage de programmation considéré ou d'un type dérivé par restriction d'un type de base. On parle de type complexe quand il s'agit d'un type obtenu par agrégation de types de base (cas des structures ou des classes). Le deuxième critère concerne le mode de transmission des paramètres avec d'une part, la transmission par valeur et d'autre part, la transmission par référence. On considère séparément ces deux modes de transmission car la nature et l'impact des erreurs sont différents.

Pour le troisième critère, nous nous basons sur les travaux de Offutt et al. [OFF93] qui définissent trois types d'usage : l'usage en calcul (Usage-C), l'usage en contrôle ou en prédicat (Usage-P), et l'usage en contrôle indirect (Usage-I). Quand on parle de l'usage de l'information partagée, on vise trois cas de figure : un paramètre d'une méthode appelée, une valeur de retour d'un appel, ou enfin une donnée ou un type de donnée partagé. Pour tous ces cas, on parlera de *donnée* dans les définitions qui suivent.

Il y a Usage-C quand une donnée est utilisée comme partie droite d'une affectation ou dans une instruction de sortie. D'une manière un peu plus formelle mais non restrictive, Usage-C peut être définit par exemple comme suit (syntaxe proche de C++, représentée dans une notation de type BNF, où → représente la dérivation et | l'alternative) :

Usage-C(x) → *droite_ affectation(x)* | *instruction_sortie(x)*
droite_ affectation(x) → *var(y) = expr(x)* | ...
expr(x) → *une expression où x apparaît*
instruction_sortie(x) → *cout << expr(x)* | ...

Par exemple *t=a*x*x-b*x ou cout << x.*

On parle d'Usage-P quand une donnée est utilisée dans un prédicat. De la même manière que Usage-C, Usage-P peut être défini comme suit :

Usage-P(x) → *conditionnelle(x)* | *iterative(x)*
conditionnelle(x) → *if predicat(x) ...* | *...*
iterative(x) → *while predicat(x) ...* | *...*
predicat(x) → *un prédicat où x apparaît*

Par exemple *if ((x*x-4*a*c)>0) ...*

On parle de Usage-I quand on a une combinaison de Usage-C et Usage-P. En d'autres termes, quand une donnée x est utilisée pour calculer une autre donnée y, et que y est ensuite utilisée dans un prédicat. Ainsi, on obtient :

Usage-I(x) → *droite_ affectation(x) ; ... ; predicat(y)*
droite_ affectation(x) → *var(y) = expr(x)* | *...*

Par exemple *t=a*x+b ; ... while (t>0) ... ;*

A ces trois type d'usage, nous rajoutons un quatrième qui est Usage-M. Il y a Usage-M quand la donnée (ou paramètre) est modifiée (ceci peut arriver notamment dans les cas de transmission de paramètres par pointeur ou par référence). En utilisant toujours la même notation, on définit Usage-M comme suit :

Usage-M(x) → *gauche_ affectation(x)* | *instruction_entree(x)* | *...*
gauche_ affectation(x) → *var(x) = expr(y)* | *...*
instruction_entree(x) → *cin >> expr(x)* | *...*

Par exemple *x=y+1 ; ...*

Dans la suite de cette section nous allons présenter les différentes formes de couplage dérivées à partir des trois critères énumérés ci-dessus.

3.2.1 Couplage dans le cas des appels de méthodes

On parle de couplage par appel de méthode quand une méthode m_1 d'une classe C_1 appelle une méthode m_2 d'une classe C_2. Dans les paragraphes qui suivent les acronymes correspondent à l'expression en anglais de la forme de couplage.

Un cas qui doit être traité à part est celui où la méthode appelée n'a pas de paramètres. Cette forme est appelée ***NPI*** .

Le deuxième cas est celui de la ***transmission des paramètres par valeurs*** ou ***le retour d'appel par valeurs.*** Dans ce cas, plusieurs scénarios sont possibles en fonction de la variation des deux critères : complexité des paramètres transmis ou de la valeur de retour (complexe vs. simple) et l'usage qui en est fait. Il est clair que nous n'avons pas une combinatoire de tous les cas. Par exemple, le mode Usage-M

n'est pas possible. De plus, nous avons décidé de ne pas considérer le cas d'un retour par valeur de type complexe. Il reste donc les cas suivants :

- ***SDI*** : Cas où un paramètre transmis est de type simple et est utilisé dans la méthode appelée en mode Usage-C.
- ***StDI*** : Cas où un paramètre transmis est de type complexe et est utilisé dans la méthode appelée en mode Usage-C.
- ***RDI*** : Cas où la valeur de retour d'un appel est utilisée dans la méthode appelante en mode Usage-C.
- ***SCI*** : Cas où un paramètre transmis est de type simple et est utilisé dans la méthode appelée en mode Usage-P.
- ***StCI*** : Cas où un paramètre transmis est de type complexe et est utilisé dans la méthode appelée en mode Usage-P.
- ***RCI*** : Cas où la valeur de retour d'un appel est utilisée dans la méthode appelante en mode Usage-P.
- ***SDCI*** : Cas où un paramètre transmis est de type simple et est utilisé dans la méthode appelée en mode Usage-I.
- ***StDCI*** : Cas où un paramètre transmis est de type complexe et est utilisé dans la méthode appelée en mode Usage-I.
- ***RDCI*** : Cas où la valeur de retour d'un appel est utilisée dans la méthode appelante en mode Usage-I.
- ***TI*** : Cas où un paramètre est transmis dans la méthode appelée à une troisième méthode avant tout autre usage. Le type de ce paramètre peut être simple ou complexe.

Le troisième cas considéré est celui où ***la transmission d'un paramètre est faite par pointeur ou par référence***. Différents cas de figure peuvent être envisagés selon la complexité du paramètre et l'usage qui en est fait dans la méthode appelée :

- ***SRDI*** : Cas où un paramètre-adresse transmis est de type simple et est utilisé dans la méthode appelée en mode Usage-C.
- ***StRDI*** : Cas où un paramètre-adresse transmis est de type complexe et est utilisé dans la méthode appelée en mode Usage-C.
- ***SRCI*** : Cas où un paramètre-adresse transmis est de type simple et est utilisé dans la méthode appelée en mode Usage-P.
- ***StRCI*** : Cas où un paramètre-adresse transmis est de type complexe et est utilisé dans la méthode appelée en mode Usage-P.
- ***SRDCI*** : Cas où un paramètre-adresse transmis est de type simple et est utilisé dans la méthode appelée en mode Usage-I.
- ***StRDCI*** : Cas où un paramètre-adresse transmis est de type complexe et est utilisé dans la méthode appelée en mode Usage-I.
- ***SRMI*** : Cas où un paramètre-adresse transmis est de type simple et est utilisé dans la méthode appelée en mode Usage-M.

- ***StRMI*** : Cas où un paramètre-adresse transmis est de type complexe et est utilisé dans la méthode appelée en mode Usage-M.

3.2.2 Couplage dans le cas des données ou type partagés

On parle de donnée partagée quand elle est définie dans une classe et utilisée dans une autre classe. C'est par exemple le cas d'une donnée membre statique dans C++, ou une variable de classe dans Smaltalk. Un type peut lui aussi être partagé, en ce sens qu'il peut être défini dans une classe et utilisé dans une autre classe. L'exemple le plus trivial est celui d'un attribut d'une classe C_1 qui a pour type une autre classe C_2. En considérant le mode d'usage, les cinq cas suivants nous ont paru les plus significatifs :

- ***GDI*** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-C.
- ***GCI*** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-P.
- ***GDCI*** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-I.
- ***GMI*** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-M.
- ***TyI*** : Cas où un type utilisateur est définie dans une classe et est utilisée dans une autre classe.

Dans cette section nous avons identifié 24 formes différentes de couplage entre classes. Pour mener à bien notre étude, il s'agit maintenant de les mesurer. La section suivante décrit la manière dont nous proposons de le faire.

3.3 Mesure du couplage entre classes

Les différentes formes de couplage entre classes identifiées dans la section précédente (appelons les CC_i où i désigne la forme) sont disjointes. Ceci veut dire que pour une classe C donnée, si on définit une forme de couplage $CC_i(C)$ comme suit :

$$CC_i(C) \subseteq (Entree(C) \cup Sortie(C))$$

on a :

$$CC_i(C) \cap CC_j(C) = \emptyset, \forall i, j$$

Pour définir une manière de mesurer les différentes formes de couplage pour chaque classe du système, nous considérons les sous ensembles $CC_i(j)$ de $CC_i(C)$ où les j sont les méthodes de la classe C. Par ailleurs, nous distinguons le couplage en entrée (cas où la classe est l'auteur de la référence externe) du couplage en sortie (cas où la classe est l'objet de la référence externe).

Donc, pour chaque classe C, on définit la mesure d'un couplage de forme k ($MCC_k(C)$) comme suit :

$$MCC_k(C) = MCCE_k(C) + MCCS_k(C)$$

où $MCCE_k(C)$ est la mesure du couplage de forme k en entrée de C et $MCCS_k(C)$ est la mesure du couplage de forme k en sortie de C. Ces deux mesures sont basées sur les mesures de couplage de forme k en entrée et en sortie pour chaque méthode j de C, et sont définie comme suit :

$$MCCE_k(C) = \sum_j MCCE_k(j)$$

et

$$MCCS_k(C) = \sum_j MCCS_k(j)$$

avec k ∈ {NPI, SDI, SCI, SDCI, StDI, StCI, StDCI, RDI, RCI, RDCI, TI, SRDI, SRCI, SRDCI, SRMI, StRDI, StRCI, StRDCI, StRMI, GDI, GCI, GDCI, GMI, TyI}

Comme les classes sont disjointes, la mesure du couplage pour une forme k ($MCC_k(S)$) est définie comme suit pour l'ensemble du système :

$$MCC_k(S) = \sum_c MCCE_k(c) = \sum_c MCCS_k(c)$$

4. Vérification de l'hypothèse

Dans cette section, nous vérifions empiriquement l'hypothèse selon laquelle certaine formes de couplage ont un impact sur la propension des classe à générer des erreurs sur un système à objets de taille moyenne. Pour cela, nous utilisons l'ensemble de mesures définies précédemment. Pour conduire cette vérification, nous utilisons dans un premier temps une méthodologie de validation de métriques de produit proposée par [BRI95]. Cette méthodologie a déjà été utilisée pour valider d'autres ensembles de mesures du produit logiciel, e.g., [BAS97]. Elle permet de déterminer sans aucune mention des valeurs le sous-ensemble de métriques pertinentes par rapport à notre hypothèse. Dans un second temps, nous utilisons une technique d'apprentissage automatique permettant d'établir un lien entre les valeurs de métriques et la caractéristique de propension. En d'autre termes, cette technique nous permet d'obtenir un modèle prédictif de la caractéristique de propension à partir de combinaisons de valeurs de métriques.

4.1 Stratégie de vérification

L'approche de vérification de notre hypothèse consiste donc à utiliser d'une part, des techniques statistiques et d'autre part, des algorithmes d'apprentissage symbolique automatique (ASA).

Nous avons utilisé la régression logistique [HOS89] pour analyser la relation entre notre ensemble de mesures de couplage et la propension des classes à engendrer des erreurs. C'est une méthode qui permet de tester l'existence de liens entre une variable que l'on cherche à expliquer (ou variable dépendante) et les variables qui expliquent le mieux l'évolution de celle-ci. Ces dernières sont appelées variables explicatives (ou variables indépendantes). Cette technique statistique a déjà été utilisé avec succès pour concevoir des modèles de qualité du logiciel et pour valider des métriques du logiciel [BRI94] [BAS96] [BRI97]. Un modèle de régression logistique multivalué est basé sur l'équation suivante :

$$p\left(X_1, \ldots, X_n\right) = \frac{e^{\left(a + \sum_i b_i * X_i\right)}}{1 + e^{\left(a + \sum_i b_i * X_i\right)}}$$

Où p représente la probabilité de l'occurrence d'une erreur dans une classe et les X_i sont les mesures de couplage considérées comme les variables explicatives du modèle.

Comme dans [BAS96] et [BRI97], nous considérons qu'une observation est la (non) détection d'une erreur dans une classe C++. Chaque (non) détection d'erreur est supposée être un événement indépendant des autres (non) détections d'erreur. Chaque vecteur de données décrit une observation et possède les composants suivants : la catégorie de l'événement (erreur ou pas d'erreur) et un ensemble de mesures caractérisant la classe étudiée. Pour chacune des mesures nous fournissons aussi les statistiques suivantes :

- Un coefficient b_i considéré comme l'estimation du coefficient de régression. Plus ce coefficient est important en valeur absolue, plus l'impact (positif ou négatif selon le signe du coefficient) de la variable explicative associée sur la probabilité p de l'occurrence d'une erreur dans la classe est important.
- Dy est basé sur la notion de ratio. Il représente la réduction ou l'accroissement du ratio y quand la variable explicative X croît d'une unité [AGR96]. Il fournit un aperçu de l'impact des variables explicatives et est plus interprétable que les coefficients de régression logistique.
- La signification statistique *p-value* qui fournit un aperçu de la précision des coefficients estimés. Elle nous indique la probabilité que le coefficient soit différent de 0 par hasard. Plus la signification statistique est importante, plus l'écart-type des coefficients estimés est important et moins notre croyance en l'impact des variables explicatives est importante.

Parallèlement, et pour générer des modèles prédictifs, nous avons exploité un algorithme d'apprentissage emprunté à la famille de l'ASA. La plupart des travaux effectués en ASA l'ont été en apprentissage supervisé. Étant donné un ensemble d'exemples (observations classifiées), les algorithmes supervisés produisent les définitions des classes auxquelles appartiennent les exemples. Pour ce faire, ils exploitent en général un langage de description attribut-valeur permettant d'avoir des considérations statistiques sur l'ensemble d'apprentissage. Deux méthodes émergent dans la famille des algorithmes utilisant le langage de description attribut-valeur, la méthode *"diviser pour conquérir"* et la méthode de *"couverture"* :

- Les algorithmes découlant de la méthode *"diviser pour conquérir"* induisent généralement une connaissance sous la forme d'arbres de décision. C'est le cas des systèmes ID3 [QUI79] [QUI86], CART [BRE84] et ASSISTANT [CES87]. Le principe de leur fonctionnement peut être résumé par l'algorithme suivant :

Si tous les exemples appartiennent à la même classe

Alors *- créer une feuille étiquetée par le nom de la classe ;*

Sinon *- Sélectionner un test basé sur un attribut (le meilleur) ;*

- Subdiviser l'ensemble d'apprentissage en sous-ensembles, chacun d'eux associés à une valeur possible de l'attribut testé ;

- Réappliquer la même procédure à chacun des sous-ensembles ainsi générés ;

Fin_du_si ;

L'étape clé de ce type d'algorithmes est le choix du "meilleur" attribut à tester afin d'obtenir des arbres de décision compactes 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.

- La deuxième méthode dite de *"couverture"* représente la connaissance induite sous la forme d'une expression logique disjonctive définissant chaque classe. AQ [MIC86] et son descendant CN2 [CLA89] appartiennent à cette famille d'algorithmes, dont l'algorithme suivant résume le principe :

- Trouver une expression conjonctive satisfaite par certains exemples de la classe ciblée et par aucun exemple des autres classes ;

- Ajouter cette conjonction comme un élément de la disjonction recherchée ;

- *Ne plus considérer les exemples satisfaisant la conjonction trouvée ; S'il reste des exemples de la classe ciblée, répéter le processus ;*

En général, ce type d'algorithmes oeuvrent à produire une définition cohérente et complète de la classe ciblée, autrement dit, une définition ne reconnaissant que les exemples de la dite classe (cohérence) et à laquelle tous les exemples adhèrent (complétude).

Nous avons choisi d'utiliser l'algorithme C4.5 [QUI93], descendant amélioré de ID3. Il permet de prendre en compte des ensembles d'apprentissage renfermant des valeurs inconnues et des attributs de types continus. Il offre aussi la possibilité de procéder à un élagage de l'arbre de décision induit et d'en dériver des règles de production. Ces améliorations permettent ainsi de considérer un éventail d'applications possibles pour C4.5 beaucoup plus vaste que ce qu'il n'est pour son ancêtre ID3.

Nous reviendrons plus en détail sur l'application de C4.5 sur nos données de couplage entre classes dans la section 4.3.

4.2 Données de validation

Dans le but de vérifier notre hypothèse, nous avons appliqué la stratégie décrite dans la section 4.1 à un environnement de développement de systèmes multi-agents appelé LALO. Ce système a été développé au Centre de Recherche Informatique de Montréal (CRIM) depuis 1993. Il comprend 87 classes C++ et possède une taille approximative de 47K lignes de code C++ (SLOC).

Dans cette étude nous avons utilisé : (1) le code source des classes C++, (2) les données sur le couplage de ces différentes classes et (3) les données relatives aux erreurs relevées dans les classes. Ces dernières correspondent à la présence concrète d'erreurs ayant été trouvées par les 50 béta-testeurs des versions 1.1 et 1.1a de LALO. La version 1.1 a été livrée en novembre 1996 alors que la version 1.1a l'a été en janvier 1997.

Les données sur le couplage des différentes classes de LALO ont été dérivées de façon statique à partir du code du système. Cela a consisté à extraire statiquement 7 faits distincts. La base de faits résultante est alors exploitée par un système de mesure à base de règles dans le but d'inférer pour chaque classe les MCC_k associées. Notons cependant que seules les classes réellement développées par l'équipe de LALO ont été considérées. En effet, les classes réutilisées à partir de librairies ou générées automatiquement par divers outils n'ont pas été prises en compte, en raison de l'impact certain qu'ont la réutilisation et les générateurs de code sur la qualité du logiciel [BAS96].

4.3 Résultats et interprétations

La table 1 consigne les statistiques descriptives des mesures de couplage extraites du système LALO. Nombre de ces mesures possèdent une variance limitée dans notre jeu de données. Ainsi, six de ces mesures n'ont aucune variance, par conséquent et dans ce jeu de données du moins, ces mesures ne peuvent être des prédicteurs de la propension des classes à engendrer des erreurs. Cette absence de variance s'explique par le fait que le système objet de l'étude est un produit OO, donc, possède des scénarios de couplage différents de ceux que l'on rencontrerait dans les systèmes procéduraux.

Table 1. *Statistiques descriptives des mesures de couplage*

Mesure	Maximum	Minimum	Moyenne	Médiane	Ecart-type
NPI	152	0	17.52	8	28.44
SDI	11	0	1	0	2.39
SCI	18	0	0	1.34	3.36
SDCI	10	0	0.29	0	1.37
RDI	88	0	9.64	4	13.83
RCI	49	0	7.24	3	10.7
RDCI	50	0	6.1	3	9.35
StDI	2	0	0.05	0	0.31
StCI	2	0	0.05	0	0.31
StDCI	2	0	0.048	0	0.308
TI	15	0	1.59	0	3.21
SRDI	19	0	0.61	0	2.32
SRCI	19	0	0.73	0	2.40
SRDCI	0	0	0	0	0
SRMI	0	0	0	0	0
StRDI	21	0	3.09	1	4.63
StRCI	21	0	2.69	1	4.75
StRDCI	11	0	0.939	0	2.19
StRMI	8	0	0.74	0	1.73
GDI	0	0	0	0	0
GCI	0	0	0	0	0
GDCI	0	0	0	0	0
GMI	0	0	0	0	0
TyI	40	0	10.65	8	8.73

Considérant les mesures GxI[2], l'application que nous avons étudiée utilise très peu de variables globales (ou données membres statiques publiques), ce qui n'est

[2] GxI : GDI, GCI, GDCI, GMI

pas le cas d'autres applications. C'est pourquoi, si nous ne pouvons valider ces mesures avec le système LALO, cela n'implique pas nécessairement qu'elles n'apporteront aucune information pertinente lors de l'analyse d'autres systèmes.

En ce qui concerne les mesures SRDCI et SRMI, toutes deux relatives à des scalaires utilisés comme paramètres échangés par les méthodes des différentes classes, il est explicable qu'elles ne possèdent pas de variance. En effet, dans des systèmes OO, les paramètres transmis sont généralement des objets et non pas des scalaires comme dans les systèmes procéduraux. Nous constatons dans la table 3 que les mesures StxI[3] ont une variance plus importante que celle de SRDCI et SRMI. Cela s'explique par le fait que les mesures StxI concernent des objets ou des structures. Des expérimentations supplémentaires sont nécessaires pour évaluer la pertinence des mesures SRDCI et SRMI, cependant, nous pensons que ce genre de mesures seraient plus pertinentes pour des systèmes procéduraux comme les mesures StxI le sont pour les systèmes OO.

La table 2 présente les mesures qui ont un réel impact sur la probabilité p qu'ont les classes C++ du système LALO d'avoir ou non des erreurs. Ne figurent dans la table que les mesures de couplage ayant une signification statistique *p-value* inférieure à 0.05. Au regard de Dy, les 8 mesures inclues dans la table 2 ont un impact considérable sur p. Ainsi, on relève un accroissement de 95.5% de y quand NPI croît d'une unité. De façon similaire, RDI, RCI, RDCI, StRDI et TyI possèdent un ratio y supérieur à 80% ce qui laisse entendre un impact important de ces mesures sur p. L'impact est moindre mais néanmoins réel pour les mesures SRDI et SRCI relatives aux scalaires.

Table 2. *Résultats de la régression logistique univariée*

Mesures	Coefficient	Dψ	p-value
NPI	-0.046	95.50%	0.007
RDI	-0.130	87.8%	<0.001
RCI	-0.092	91.21%	0.0003
RDCI	-0.142	86.76%	0.0003
SRDI	-1.195	30.27%	0.001
SRCI	-1.160	31.34%	0.001
StRDI	-0.106	89.94%	0.012
TyI	-0.198	82.04%	<0.001

Pour compléter notre étude et mieux comprendre la relation entre le couplage au niveau code et la qualité du logiciel, nous avons construit un modèle exploitable pour évaluer à priori la propension des classes d'un système OO à engendrer des erreurs, en raison de leur niveau de couplage. Pour ce faire, nous avons eu recours à l'algorithme d'apprentissage supervisé C4.5 [QUI93] qui induit un modèle de

[3] StxI : StRDI, StRCI, StRDCI, StRMI.

classification sous la forme d'un arbre de décision ou de règles à partir d'un ensemble d'exemples. Chaque exemple (i.e., classe C++ du système LALO) de l'ensemble d'apprentissage possède la même structure : un ensemble de paires attribut-valeur décrivant l'exemple, dans notre cas, les différentes mesures de couplage associées à la classe C++. Un attribut particulier représente la classe de l'exemple, c'est-à-dire, le fait d'avoir relevé ou non des erreurs dans la classe C++. Le choix de C4.5 découle de son utilisation aisée et rapide ; de plus, les résultats générés sont faciles à interpréter et l'ensemble d'apprentissage facile à construire. Une motivation majeure vient aussi du fait que nous sommes familiers avec cet algorithme qui a déjà été exploité dans des travaux antérieurs pour construire des modèles prédictifs de la qualité du logiciel [BAS97]. En fait, les modèles générés par ce type d'algorithme peuvent être considérés comme complémentaires à ceux provenant des techniques statistiques. Ils seront utiles à ceux non familiers avec l'approche statistique.

Le problème est donc le suivant : déterminer un modèle de classification qui répondant à des questions sur les valeurs des mesures de couplage d'une classe C++, prédit correctement la propension de cette classe à avoir des erreurs. Dans notre exemple et pour la préparation de l'ensemble d'apprentissage, les classes du système LALO ayant eu plus d'une erreur se sont vu attribué la valeur *"oui"* pour l'attribut classe *erreurs* tandis que les autres recevaient la valeur *"non"*.

Une fois le modèle de classification induit par C4.5, il s'agit de l'évaluer pour juger de son efficacité prédictive. Si le modèle obtenu possède une bonne efficacité prédictive, cela implique que nos métriques de couplage sont pertinentes pour identifier des classes susceptibles d'engendrer des erreurs. Deux critères sont mesurés : la cohérence et la complétude du modèle. La cohérence est définie comme le pourcentage de classes C++ jugées propices aux erreurs et qui le sont réellement. Cette grandeur doit être maximisée sinon, cela impliquerait que le modèle identifie des classes comme étant propices aux erreurs sans qu'elles le soient réellement. Cela peut mener à une allocation démesurée de ressources pour vérifier ces classes. Quant à la complétude, elle représente le pourcentage de classes C++ réellement propices aux erreurs et qui ont été jugées comme telles. Dans ce cas aussi, il s'agit de maximiser la complétude, car dans le cas contraire, on sous-estimerait la quantité de ressources à allouer à la vérification des dites classes.

Le procédé de calcul de ces deux grandeurs s'appuie sur une procédure dite *de "V-fold cross-validation"* [BRE84], souvent utilisée lorsque les exemples d'apprentissage ne sont nombreux. Elle consiste à générer pour chaque exemple X un modèle de classification à partir du restant des exemples (ensemble des exemples - X). Ce modèle est alors utilisé pour classifier X. La table 3 consigne les résultats quantitatifs obtenus avec C4.5. Ceux-ci laissent apparaître un bon comportement de C4.5, lorsqu'il est confronté à ce type d'expérimentations.

Table 3. *Résultats quantitatifs obtenus à l'aide de C4.5*

Valeur réelle de l'attribut "erreurs"	Valeur trouvée par le biais du modèle *non*	*oui*	Complétude
non	47	7	**87**
oui	11	20	**64**
Cohérence	**81**	**74**	

Cohérence globale	**78.82%**
Incohérence globale	**21.18%**

Le modèle de classification généré par C4.5 consiste en 5 règles (4 règles explicites et 1 règle par défaut). Ces règles sont :

Règle 0 : SRCI > 0
TYI > 7
→ erreurs **oui** [89.9%]

Règle 1 : NPI > 16
RDCI > 0
StRMI <= 3
→ erreurs **oui** [79.5%]

Règle 2 : SDI <= 0
StRCI > 2
StRMI <= 3
→ erreurs **oui** [64.5%]

Règle 3 : NPI <= 16
SRCI <= 0
→ erreurs **non** [80.2%]

Classe par défaut : erreurs **non**

Si nous considérons par exemple la règle 1 du modèle, nous pouvons la traduire par :

> *"Une classe est propice à engendrer des erreurs, si la mesure de couplage NPI est supérieure à 16, la mesure RDCI est non nulle et la mesure StRMI est inférieure ou égale à 3".*

Il demeure que les règles ainsi obtenues sont relativement spécifiques à l'équipe de développement du système LALO. Confrontées à un système d'un tout autre type, elles n'apporteraient peut-être aucune information pertinente à l'équipe de développement. Idéalement, il faudrait disposer du code de nombreux systèmes tels que LALO, pour pouvoir en extraire des règles de prédiction d'un degré de généralité plus élevé.

Nous avons aussi exploré la pertinence que pouvait avoir les mesures de couplage en entrée et en sortie, telles qu'elles ont été définies dans la section 3.3. Pour ce faire, nous avons procédé comme précédemment, si ce n'est que l'ensemble d'apprentissage change à chaque fois : (1) avec seulement les mesures de couplage en entrée et (2) avec uniquement les mesures de couplage en sortie. Les performances des deux modèles ainsi obtenus sont consignées dans les tables 4 et 5.

Table 6. *Modèle prédictif induit à partir des mesures de couplage en entrée*

Valeur réelle de l'attribut "erreurs"	Valeur trouvée par le biais du modèle *non*	*oui*	Complétude
non	50	4	**92**
oui	8	23	**74**
Cohérence	**86**	**85**	

Cohérence globale	**85.88%**
Incohérence globale	**14.12%**

Table 5. *Modèle prédictif induit à partir des mesures de couplage en sortie*

Valeur réelle de l'attribut "erreurs"	Valeur trouvée par le biais du modèle *non*	*oui*	Complétude
non	45	9	**83**
oui	17	14	**45**
Cohérence	**72**	**60**	

Cohérence globale	**69.41%**
Incohérence globale	**30.59%**

Il découle de ces résultats que le modèle basé uniquement sur les mesures de couplage en entrée possède les meilleures performances, en termes de cohérence et de complétude. Ce résultat démontre que les classes sont plus vulnérables aux

changements effectués dans les classes qui leur sont couplées. Une conséquence de ce résultat est que nous pouvons nous contenter d'utiliser le modèle de la table 4 pour prédire quelle classe est propice à engendrer des erreurs, ce qui réduirait la tâche d'analyse du code source, en diminuant de moitié le nombre de mesures à collecter. Cependant, les mesures de couplage en sortie semblent avoir un potentiel d'identification des classes candidates à être réutilisées. En effet, les classes ayant des mesures de couplage en sortie élevées devraient être analysées plus finement pour éventuellement les transformer en composants d'une librairie de classes réutilisables. Cet axe de recherche fait partie de nos travaux actuels.

5. Conclusion

Dans cet article nous décrivons une expérience de vérification d'une hypothèse de corrélation d'une caractéristique de qualité (propension d'une classe à engendrer des erreur) avec des attributs quantitatifs d'un logiciel (métriques de couplage). Pour mener à bien cette expérimentation, nous avons défini 24 formes de couplage entre deux classes. Pour chaque forme, nous distinguons les cas d'importations (cas où la classe est l'auteur de la référence) des cas d'exportations (cas où la classe est l'objet de la référence). Nous avons ensuite proposé une manière de mesurer ces types de couplage. Pour déterminer parmi ces formes de couplage celles qui ont une influence sur la propension à générer des erreurs, nous avons utilisé une méthodologie de validation statistique. Nous avons aussi bâti un modèle prédictif. L'approche que nous avons suivie est celle de l'apprentissage symbolique supervisé à travers l'algorithme C4.5. Nous avons utilisé un système de 87 classes codées en C++ pour extraire les métriques. Pour calculer l'exactitude et la complétude du modèle obtenu, nous avons utilisé une technique de validation croisée qui consiste, pour chaque exemple X, à apprendre à partir des autres exemples (échantillon - X), puis à prédire X. Cette forme de validation est généralement utilisée quand l'ensemble des données d'apprentissage est relativement petit.

Les résultats de nos expérimentations ont démontré que nos mesures peuvent prédirent avec un niveau élevé d'exactitude quelles classes peuvent générer des erreurs. Les trois modèles prédictifs obtenus (couplage global, couplage en entrée et couplage en sortie) ont donné des résultats très satisfaisants (92% de complétude par exemple).

L'utilité d'un modèle de prédiction est multiple. Il peut être utilisé pour déterminer quelles classes doivent être particulièrement testées et inspectées. Dans le cas de la maintenance préventive, il peut déterminer quelles classes doivent être réécrites et de quelle manière. Il peut enfin être utilisé pour détecter les composants potentiellement réutilisables dans une application écrite, sans souci de réutilisation.

Dans une approche similaire par la méthodologie mais néanmoins différente dans ses besoins et buts, nous avons été amenés à utiliser un autre algorithme d'apprentissage au pouvoir expressif plus important [ALM98]. Il s'agit de FOIL [QUI90], dans le langage de description des exemples et de la connaissance induite (clauses) est d'ordre 1. Il possède de ce fait un meilleur pouvoir expressif que les algorithmes basés sur le langage attribut-valeur, et permet entre autres de découvrir

des relations entre les variables indépendantes (i.e., les métriques) qui lui sont soumises. De plus, FOIL a donné de meilleurs résultats quantitatifs que C4.5. Cette famille d'algorithmes fera l'objet d'investigation supplémentaire dans le cadre de nos travaux.

6. Bibliographie

[AGR96] A. Agresti. "*An Introduction to Categorical Data Analysis"* Wiley Series in Probability and Statistics, 1996.

[ALM98] M. A. De Almeida, H. Lounis, and W. L. Melo. "An Investigation on the Use of Machine Learned Models for Estimating Correction Costs". A paraître dans *Proc. of the 20th IEEE International Conference on Software Engineering, Lessons and Status Reports ICSE'98-LSR*, 1998.

[BAS96] V. Basili, L. Briand, and W. Melo. "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Trans. on Software Engineering*, vol 22, no. 10, October, 1996 pp-751-761.

[BAS97] V. Basili, S. E. Condon, K. El Emam, R. Hendrick, and W. L. Melo. "Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components", In *Proc. of the 19th Int'l Conf. on Software Engineering*, 1997.

[BEA94] F. Brito e Abreu and R. Carapuça. *"Candidate metrics for object-oriented software within a taxonomy framework". Journal of System and Software*, 26(1):87–96, 1994.

[BOO94] G. Booch. *"OO Analysis and design with applications"*. 2nd edition, Benjamin Cunnings, 1994.

[BRE84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *"Classification and Regression Tress"*, Published by Wadsworth, 1984.

[BRI94] L. Briand, S. Morasca, V. Basili. "Defining and Validating High-Level Design Metrics", Technical Report no. CS-TR-3301, University of Maryland, Dep. of Computer Science, 1994.

[BRI95] L. Briand, K. El Emam, S. Morasca. "Theoretical and Empirical Validation of Software Product Measures". ISERN technical report 95-03, 1995.

[BRI96] L. C. Briand, S. Morasca, and V. R. Basili. "Property-Based Software Engineering Measurement". *IEEE Trans. on Software Engineering*, 22(1), 68-85, 1996.

[BRI97] L. Briand, P. Devanbu, and W. L. Melo. "An Investigation into Coupling Measures for C++". *In Proc. of the 19th Int'l Conf. on Software Engineering*, 1997.

[CES87] B. Cestnik, I. Bratko, I. Kononenko. "ASSISTANT 86: a knowledge elicitation tool for sophisticated users". *Progress in machine learning*, Sigma Press, 1987.

[CHI94] S.R. Chidamber and C.F. Kemerer: "A Metrics suite for Object Oriented Design". *IEEE Transactions on Software Engineering*, 20(6):476-493, June 1994.

[CLA89] P. Clark & T. Niblet. "The CN2 induction algorithm". In *Machine Learning Journal*, vol. 3, 1989, p 261-283

[CON79] Constantine and E. Yourdon. *"Structured Design"*. Prentice-Hall, Englewood Cliffs, NJ, 1979.

[DAV96] Prem Devanbu, Sakke Karstu, Walcelio Melo and William Thomas. "Analytical and Empirical Evaluation of Software Reuse Metrics". In *Proc. of the 18th Int'l Conf. On Software Engineering*, 1996 p-189-199.

[HOS89] D. Hosmer and S. Lemeshow. *"Applied Logistic Regression"*. Wiley-Interscience. 1989.

[LOU97] H. Lounis and W. L. Melo. "Identifying Coupling in Modular Systems". In *Proc. of the 7th Int'l Conf. nn Software Technology*, 1997, p 23-40.

[MIC86] R.S. Michalski, I. Mozetic, J. Hong, & N. Lavrac. “The AQ15 inductive learning system: an overview and experiments”. Technical report UIUCDCS-R-86-1260, dpt of computer science, university of illinois at Urbana-Champaign, 1986

[OFF93] A. J. Offutt, M. J. Harrold, and P. Kolte. "A Software Metric System for Module Coupling". *Journal on Software and System*, 1993.

[PAG80] M. Page-Jones. *"The Practical Guide to Structured Systems Design"*. Yourdon Press, New York, NY, 1980.

[PRI97] M. W. Price and S. A. Demurjian, Sr. "Analyzing and Measuring Reusability in Object-Oriented Design", *Proc. of OOPSLA'97*, 1997.

[QUI79] J.R. Quinlan. “Discovering rules from large collections of examples: a case study”. In *E.S in the micro-electronic age*, D.Michie (Ed), Edinburgh university press, 1979.

[QUI86] J.R. Quinlan. “Induction of decision tree”. *Machine Learning journal* 1, p 81-106, 1986.

[QUI90] J.R. Quinlan. “Learning Logical Definitions from Relations”. In *machine learning journal*, vol 5, n°3, p 239-266, August 1990.

[QUI93] : J. R. Quinlan. *"C4.5: Programs for Machine Learning"*. Morgan Kaufmann Publishers, Sao Mateo, CA, 1993.

[SOM95] I. Sommerville. *"Software Engineering"*. Addison-Wesley, fifth edition, 1995.