Université de Montréal

Extension des modèles de prédiction de la qualité du logiciel en utilisant la logique floue et les heuristiques du domaine

par

Mohamed Adel SERHANI

Département d'Informatique et de Recherche Opérationnelle
Faculté des Arts et des Sciences

Mémoire présenté à la Faculté des Études Supérieures
en vue de l'obtention du grade de
Maître ès Sciences (M.Sc.)
en Informatique

Décembre, 2001

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :

Extension des modèles de prédiction de la qualité du logiciel en utilisant la logique floue et les heuristiques du domaine

présenté par :

Mohamed Adel SERHANI

a été évalué par un jury composé des personnes suivantes :

Claude FRASSON
Président-rapporteur

Houari A. SAHRAOUI
Directeur de recherche

Mounir BOUKADOUM
Codirecteur de recherche

Julie VACHON
Membre du jury

Mémoire accepté le 17 février 2002

Sommaire

La nécessité de construire et de comprendre des modèles prédictifs pour améliorer la qualité de logiciel de manière précise et explicite, indépendamment de tous critères (Ex : échantillon de données) nous amène à chercher une nouvelle approche pour contourner les problèmes des techniques existantes de construction et d'utilisation des modèles de prédiction de la qualité du logiciel.

Dans cette thèse, nous proposons une approche hybride liant deux familles de travaux pour la construction et l'utilisation des modèles de prédiction de la qualité de logiciel. Ces deux familles offrent des avantages complémentaires : d'une part les approches basées sur les données historiques dérivent statistiquement des rapports corrects entre les attributs internes de logiciel (métriques) et les caractéristiques internes de la qualité. D'autre part, les approches basées sur les connaissances du domaine sont faciles à utiliser grâce à leurs capacités explicatives. Plutôt que de construire un nouveau modèle à partir de rien (d'aucune base), nous proposons d'étendre l'ancien modèle en utilisant les heuristiques du domaine. Afin de permettre cette extension, nous transformons les valeurs seuils précises des règles du modèle naïf en utilisant la logique floue.

Le modèle hybride proposé a pour objectif de :

- Résoudre le problème d'utilisation des valeurs seuils associées aux modèles d'évaluation en les remplaçant par des valeurs seuils floues (identification des tendances en incluant d'autres causes).
- Améliorer l'exactitude d'évaluation en incluant les heuristiques du domaine
- Faire une validation empirique et inclure les jugements des experts

Nous présentons une approche pour étendre les modèles basés sur les règles classiques de prédiction en intégrant les connaissances du domaine. Le but de l'extension est de transformer les modèles naïfs en modèles causals aidant dans le processus de prise de décision. Nous décrivons comment, à partir d'un système basé sur les règles classiques,

nous pouvons détourner ses faiblesses, à savoir l'utilisation des valeurs seuils numériques et les limites d'utilisation des règles dérivées pour supporter l'aide à la décision, en utilisant la logique floue et en ajoutant les heuristiques du domaine.

Nous avons également conduit une expérimentation pour comparer le modèle initial et le modèle étendu afin de valider notre approche. Dans cette expérience, nous avons étendu un modèle de prédiction de l'instabilité existant en ajoutant les connaissances du domaine. L'expérience a prouvé que le modèle étendu améliore de manière significative l'exactitude de prédiction. Elle prouve également que les utilisateurs apprécient les capacités explicatives offertes par le modèle étendu.

Mots Clés : *modèle causal, modèle naïf, fuzzification, defuzzification, métriques de logiciels, logique floue, qualité de logiciel, modèle prédictif.*

Abstract

The need to build and understand predictive model to improve software quality in a precise and explicit manner, independently of any criteria (sample data) brings us to look for a new approach to circumvent the problems with existing models to build and apply software quality estimation models.

In this thesis, we propose a hybrid approach to bridge the gap between two families of work for building and using software quality prediction models. The two families offer complementary advantages. Historical measurement data-based approaches derive statistically correct relationships between software internal attributes metrics and quality characteristics. In the same time, domain knowledge-based approaches are easy to use for their explanatory capabilities. Rather than building the latter from scratch, we propose to extend the former using domain heuristics. To allow this extension, we first transform the precise threshold values of naïve model rules using fuzzy logic.

Our proposed hybrid model can handle:

- The problem of using precise threshold values associated with the estimation models by replacing them with fuzzy threshold values (identification of the trends by including others causes).
- The improvement of estimation accuracy by including domain specific knowledge
- Empirical validation and expert judgment

We present an approach to extend classical rule-based prediction models by integrating domain knowledge. The goal of the extension is to transform naïve models into more causal models that are good supports for decision-making. We describe how, starting from a classical rule-based system, we can circumvent its weaknesses, namely the use of numerical threshold values and the limited usefulness of the derived rules for decision support, by using fuzzy logic and by adding domain heuristics.

We also conducted an experiment to compare the original and the extended models in order to validate our approach. In this experiment, we extended an existing instability prediction model by adding domain knowledge. The experiment showed that the extended model significantly improves the correctness of the predictions process. It also shows that the users appreciate the explanatory capabilities offered by the extended model.

Keywords: *causal model, naïve model, fuzzification, defuzzification, software metrics, fuzzy logic, software quality, predictive models.*

Remerciements

Je tiens à remercier vivement le professeur Houari Sahraoui, mon directeur de recherche, pour m'avoir encadré dans mon sujet de thèse, pour sa disponibilité, ses directives, ses conseils, son soutien technique et son appui financier.

Je tiens à remercier également le professeur Mounir Boukadoum, mon codirecteur de recherche, pour avoir suivi mon travail et pour m'avoir donné ses conseils tout au long de cette maîtrise.

Je me tiens à cœur de remercier plus profondément mes chers parents qui ont fait de moi ce que je suis aujourd'hui. Que dieu, le tout puissant, leur donne santé et longue vie.

Des remerciements particuliers à ma femme Amal pour m'avoir soutenu durant mon travail.

Mes remerciements aussi à mes frères, sœurs grands parents et oncles pour le liens très intimes qu ont as établis entre nous. Que tous les membres de ma famille trouvent ici la reconnaissance des soutiens qu'ils m'ont toujours apportés.

Je remercie encore tous mes amis de l'université membres du groupe Gélo et des amis de l'extérieur de l'université pour leurs encouragements et pour le climat familial et chaleureux dans lequel on a vécu ensemble, tout le temps.

J'aimerais remercier tous les membres de l'équipe GLIC du CRIM pour leur participation et leur dialogue constructif.

Finalement, je remercie toutes les personnes qui ont contribué directement ou indirectement à la réalisation de ce travail.

Table des matières

Liste des figures

Liste des tableaux

Liste des sigles et abréviations

ISO : International Standard Organisation
OO1 : Outils de prédiction d'évaluation et de restructuration de la qualité

NASA : National Aeronautics and Space Administration
DISCOVER : Environnement d'analyse de code et d'évaluation de la qualité
MI : Moteur d'inférence
FA : Fonction d'Appartenance

BNF : Backus Naur Form

OO : Orienté Objet

API : Application Programming Interface

UML : Unified Modeling langage

XML : eXtensible Markup Language

Introduction et problématique

L'avènement des Normes ISO 9000, ISO/SPICE et, plus généralement, la prise de conscience du besoin en matière de qualité des logiciels et des systèmes ont poussé les entreprises à chercher des solutions visant à l'amélioration de leurs pratiques et à l'optimisation des processus de développement et de maintenance des logiciels. Cette prise de conscience s'est accélérée avec la formalisation des concepts et des pratiques rattachées à la qualité.

Aujourd'hui, des normes internationales et des standards éprouvés existent et fournissent un cadre d'expression et d'action. Désormais, la qualité n'est plus une volonté qui se décrit en marge de l'objet auquel elle s'applique; elle fait partie de l'objet lui-même. Depuis longtemps déjà, certaines entreprises, voulant assurer à leurs clients un certain niveau de qualité des "Systèmes" qu'elles fabriquent, se sont données les moyens d'intégrer dans leur processus de développement la dimension du contrôle de la qualité au niveau logiciel. C'est notamment le cas des industries de l'aéronautique, de l'espace, du transport, de l'armement et du monde médical.

Quantifier la qualité est important. Cela facilite pour l'acheteur le processus de prise de décision. En général, les mesures quantitatives de la qualité permettent de prévoir la satisfaction que l'on pourrait tirer d'un logiciel avant d'avoir la chance de l'utiliser pratiquement. Cela est utile quand on ne connaît pas le logiciel ou lorsque celui-ci est en cours de développement.

Cependant, la définition de la qualité pose un grand problème: beaucoup de gens penseront que la qualité est simplement l'absence des erreurs. Les définitions des dictionnaires sont très vagues. La meilleure, peut être, est celle donnée par le

dictionnaire *Oxford English Dictionary* selon lequel la qualité est vue comme un caractère particulier de supériorité. Plusieurs références en génie logiciel définissent la qualité comme étant l'implémentation correcte des spécifications. Une telle définition est utile lors du processus de développement mais ne permet pas à l'utilisateur final de comparer entre plusieurs approches. Selon l'ISO, la qualité est l'ensemble des caractéristiques d'un produit ou d'un service qui porte sur sa capacité à satisfaire des besoins implicites ou explicites. L'IEEE définit la qualité par le degré avec lequel un système, un composant ou un processus possède une combinaison d'attributs désirés. Pressman définit la qualité dans [1] comme la conformité aux exigences explicites, à la fois fonctionnelles et de performances, aux standards de développements explicitement documentés et aux caractéristiques implicites qui sont attendues de tous logiciels professionnellement développés. Cependant, des utilisateurs différents peuvent avoir des attentes différentes d'un même produit. Par exemple, un débutant est plus concerné par la facilité d'usage et d'apprentissage, aux dépens des performances; un administrateur de système peut être plus intéressé par la détection automatique des erreurs et le recouvrement sur erreur que par la facilité d'installation, etc.

La notion de subjectivité dans la définition de la qualité peut être contournée en étudiant chacune des caractéristiques à part (ex : stabilité), cela nous permettant de rendre moins abstraite la définition de la qualité et de lier son champ de définition à plusieurs critères constituant ensemble la notion concrète de la qualité.

Pendant une longue période, de nombreuses métriques ont été inventées pour mesurer la qualité du logiciel. Cependant, quantifier la qualité suppose un accord universel sur ce que constitue la qualité. Elle a donc été considérée comme une qualité absolue sujette à des mesures objectives. Or, la qualité est comme la beauté, sa valeur dépend de l'œil qui est entrain de la juger; n'importe quelle mesure de qualité véhicule beaucoup de subjectivité.

La qualité est généralement définie par l'aptitude d'un produit à atteindre ses spécifications. Elle a été vue comme un ensemble de caractéristiques; ces dernières à

leur tour sont définies par des ensembles de critères auxquels on peut associer des indicateurs mesurables. Dans ce contexte, prédire la qualité revient à mesurer ces caractéristiques externes, à savoir la stabilité, la réutilisabilité, la compréhensibilité, etc. Cependant, ces dernières sont difficilement mesurables à priori et doivent l'être à l'usage.

Les contributions majeures de ce projet seront les suivantes:

- Bâtir un modèle de prédiction de la qualité pour la stabilité des systèmes à objets.
- Identifier les métriques indicatrices de stabilité.
- Introduire la logique floue pour gérer l'incertitude inhérente à la prédiction.
- Généraliser l'application du modèle en évitant le problème des valeurs seuils.
- Faciliter la prise des décisions et gérer le risque lié à plusieurs causes en utilisant un modèle causal.
- Faciliter la validation par des experts des modèles prédictifs et donc leur acceptation à grande échelle.

Contexte et problématique

Les considérations de la qualité sont de plus en plus devenues d'importance primordiale dans le processus de développement et de conception d'un logiciel de qualité, tandis que la plupart des logiciels sont conçus sans prise en compte explicite de leur stabilité. Cependant, le passage d'une version de logiciel à l'autre nécessite beaucoup d'effort (ajout de nouvelles fonctionnalités, correction des erreurs, adaptation aux nouvelles plates-formes, etc..). Plusieurs études ont montré que 65 % à 75 % de l'effort total déployé durant le cycle de vie d'un logiciel l'est, en fait, pour sa maintenance. Plus de 50% de l'effort fourni en maintenance perfective et corrective est dépensé à comprendre les programmes existants. La plupart des techniques utilisées pour bâtir des modèles prédictifs de qualité de logiciel présentent plusieurs défauts de conception ou de précision. Cependant, d'autres critères doivent être pris en compte dans le processus de

prédiction tel que la compréhensibilité, la facilité d'utilisation et l'exactitude des résultats obtenus.

Notre travail se base sur l'étude d'une caractéristique de la qualité : **la stabilité**. Le but ultime de notre approche est de proposer un modèle pour prédire la stabilité des systèmes à objet, de répondre à la question :

Qu'est ce qui peut influencer l'évolution d'une classe ?

La définition de la stabilité selon ISO9126 correspond au nombre d'éléments du logiciel qui peuvent entraîner un risque d'effets non souhaités lors d'une modification : un logiciel est stable si sa modification n'a pas d'impact sur les autres logiciels ou en a peu.

Solution proposée

L'idée générale pour estimer la qualité d'un logiciel est d'utiliser des attributs internes du logiciel (Ex : hiérarchie, couplage, cohésion, complexité, etc..) qui sont directement mesurables, mais difficilement interprétables. Ces derniers ont une grande utilité s'ils sont reliés à des caractéristiques externes de qualité, c'est-à-dire si on peut les utiliser pour prédire des caractéristiques de la qualité. Mieux encore, ils pourraient servir à établir un lien de cause à effet entre les deux.

Notre approche a pour objectif de définir un modèle de prédiction de la stabilité des bibliothèques de classes à objets. Cela permet de vérifier si la conception d'une bibliothèque ne va pas entraver son évolution. Nous proposons une solution basée sur un modèle causal permettant de capturer une tendance plutôt que des valeurs seuils et d'expliquer les relations qu'il y a entre la stabilité et les facteurs d'influence (métriques).

L'idée est d'utiliser la logique floue pour avoir des seuils flous et un modèle causal pour intégrer les heuristiques du domaine afin d'expliquer les relations de causalité (métriques-stabilité). Nous bâtissons ces modèles à partir des métriques du logiciel (indicateurs de la qualité) et d'une nomenclature de changement (évolution/stabilité) à

définir. Une expérimentation aura lieu sur des échantillons des bibliothèques avec différentes versions.

Plan du mémoire

Le chapitre 2 introduit les concepts de base des différentes techniques de prédiction de la qualité du logiciel. Nous donnons une description de chaque approche et nous présentons à la fin une synthèse globale basée sur une étude comparative entre ces techniques.

Le chapitre 3 donne un aperçu général sur la logique floue. Nous définissons le principe général de cette dernière et nous exposons les principales étapes de la conception floue. Ensuite, nous abordons nos motivations du choix de la logique floue pour modéliser notre système. Enfin, nous présentons les champs d'application de la logique floue dans l'industrie.

Le chapitre 4 décrit notre approche de construction d'un modèle prédictif en traçant les objectifs et les motivations liées à cette approche. Ensuite, nous définissons la caractéristique de qualité à prédire et les métriques liées à cette dernière. Nous présentons finalement le modèle naïf sous forme de règles estimant la stabilité.

Le chapitre 5 met en évidence les principes théoriques mentionnés dans le chapitre précédent. Il décrit en détails les étapes de fuzzification, de dérivation des règles floues, d'inférence des règles et de défuzzification.

Le chapitre 6 décrit les parties implémentation de ce projet. Nous motivons nos choix des solutions utilisées pour modéliser et représenter notre système. Nous présentons ensuite l'architecture générale du système en fonctionnalités et en modules. Nous présentons chaque module du système et expliquons les diagrammes de classes correspondants. Nous décrivons à la fin l'outil OO1, surtout les fonctions prédiction et restructuration, et nous proposons les améliorations futures de l'outil.

Le chapitre 7 sert à valider notre approche. Nous expliquons au début le principe de validation ensuite nous présentons les objectifs de notre expérimentation et la méthodologie suivie pour l'effectuer; nous utilisons l'outil OO1 pour valider nos modèles de prédiction. Enfin nous faisons une analyse et une synthèse des résultats obtenus.

Nous terminons enfin cette étude par une conclusion portant sur nos travaux et une série d'orientations pour les travaux à venir.

Modèles prédictifs de la qualité du logiciel

Dans ce chapitre, nous effectuons un tour d'horizon des principaux modèles utilisés pour la prédiction de la qualité de logiciel. Nous illustrons leur fonctionnement par un exemple explicatif de chaque modèle. Ensuite, nous faisons une synthèse générale en présentant un tableau comparatif entre modèles. Finalement, nous donnons une conclusion en soulignant les éléments manquants dans un modèle prédictif, en expliquant la différence entre un modèle naïf et un modèle causal.

2.1 GÉNÉRALITÉS

Construire un modèle de prédiction revient généralement à établir une relation de cause à effet entre deux types de caractéristiques du logiciel : les attributs internes qui sont directement mesurables tels que la taille, l'héritage, le couplage, et des caractéristiques de qualité qui sont mesurables après un certain temps telles que la maintenabilité, la stabilité et la réutilisabilité.

Dans la communauté de recherche en génie logiciel, plusieurs modèles ont été construits pour prédire la réutilisabilité, la maintenabilité et d'autres caractéristiques de qualités des composantes orientées objet. On peux citer les travaux de Basili et al [2] pour prédire la prédisposition aux fautes "faults proneness", les travaux de Demeyer et Ducasse [3] pour prédire la stabilité des cadres d'applications, les travaux de Price et Demurjian [4] pour prédire la réutilisabilité des classes à objets, les travaux de Li et Henry [5] pour prédire l'effort de maintenance, et les travaux de Sahraoui et al qui exploitent dans [6,7,8] les techniques statistiques et d'apprentissage pour estimer la réutilisabilité, et la prédisposition aux fautes des bibliothèques à objets. Ces modèles diffèrent selon au moins deux points de vues:

- La technique qu'ils adoptent pour construire le modèle de la qualité : certains d'entre eux utilisant des techniques statistiques, d'autres se basant sur des techniques d'apprentissage ou d'arbre de décision.
- Le principe de raisonnement de chaque modèle.

Plusieurs groupes de recherche exploitent ou combinent des techniques (méthodes ou approches) pour construire des modèles prédictifs. On retrouve des techniques statistiques, des techniques à réseau bayésien, des techniques à arbre de décision, des techniques à logique floue, des techniques à base de règles ou à base de cas et des techniques basées sur les réseaux de neurones. Des approches hybrides qui combinent deux techniques pour construire un modèle prédictif peuvent aussi être utilisées (Ex : technique hybride neuro-floue). Tous ces groupes ont des objectifs en commun qui sont, entre autres, la dérivation des procédures permettant:

- de manipuler une large variété de problèmes représentés par des données suffisantes, ce qui permet de faciliter la généralisation des résultats obtenus.
- leur applicabilité dans un contexte purement expérimental prouvant leur succès.
- d'avoir un comportement décisionnel similaire à celui de l'être humain, mais ayant l'avantage de l'uniformité et de l'explicité.

2.2 QUELQUES DEFINITIONS

Métrique: est une assignation empirique objective d'un nombre ou d'un symbole a une entité pour caractériser un attribut spécifique et le décrire selon des règles clairement définies [9].
Exemple : DIT décrit l'attribut profondeur dans un arbre d'héritage de l'entité classe.

Caractéristique de qualité: ensemble d'attributs portant sur le rapport existant entre le niveau de service d'un logiciel et la qualité de ressources utilisées, dans des conditions déterminées [9].

Exemple : maintenabilité, fiabilité, etc. On peut trouver leurs définitions dans la norme ISO/IEC 9126.

Modèle prédictif: consiste en un modèle mathématique et d'un ensemble de procédure de prédiction pour déterminer des paramètres inconnus et interpréter des résultats [9].
Exemple : COCOMO modèle de prédiction permettant de prédire l'effort et la charge du développement en fonction de la taille du code.

Modèle naïf: est un modèle prédictif se traduisant par des relations directe entre des attributs prédictifs et la caractéristique de qualité prédite [10].
Exemple : Effort = f (taille)

Modèle causal: est un modèle prédictif se traduisant par une succession de relations de causalité impliquant d'une part les attributs prédictifs et les conclusions intermédiaires avec la caractéristique prédite. Chacune de ces relations et intuitive (voir l'exemple de la page 10)

2.3 APPROCHES STATISTIQUES

Plusieurs travaux de recherche dans le domaine de la qualité ont été basés sur l'utilisation des techniques statistiques pour prédire la qualité d'un logiciel. Les travaux présentés par Briand et al [3,4,5,6] sont les plus prometteurs dans ce domaine. Ces derniers utilisent des techniques de régression simple et à variables multiples qui permettent de prédire l'effort de maintenance à l'aide des métriques collectées à partir du code source.

La forme d'un modèle prédictif basée sur une régression à variables multiples est comme suit :

$$\pi(X_1, L, X_n) = \frac{e^{(c_0+c_1X_1+L+c_n)}}{1+e^{(c_0+c_1X_1+L+c_nX_n)}}$$

(Le modèle de régression simple diffère par l'existence d'une seule variable indépendante)

π est la variable dépendante; elle représente la probabilité de trouver une faute dans une classe, les X_i sont les variables indépendantes (mesures de qualité : les métriques). C_0 C_n sont des coefficients qui évaluent l'impact des variables indépendantes sur la variable dépendante.

Pour illustrer ce concept, considérons l'exemple d'un modèle de prédiction basée sur la régression à une seule variable pour prédire les prédispositions aux fautes dans une bibliothèque de classes. La variable indépendante est la métrique DIT (profondeur dans l'arbre d'héritage). Nous appliquons la formule pour une valeur de DIT 3 le résultat est : π (3) = 0.4, nous interprétons ce résultat comme suit : " il y a 40% de probabilité pour détecter les fautes dans la classe ayant un DIT=3 " ou " 40% des classes qui ont un DIT est égal à 3 contient des fautes "

2.4 APPROCHES PAR RÉSEAU BAYÉSIEN

Un réseau bayésien est un graphe dont les nœuds définissent les variables du système et dont les arcs définissent l'existence de relations entre ces variables. Les deux notions essentielles que modélisent les réseaux bayésiens sont l'incertitude et la causalité. La notion d'incertitude est présente dans les nœuds du réseau qui sont des variables aléatoires. La notion de causalité, quant à elle, provient des directions des arcs liant les nœuds du réseau. Plus formellement, ces arcs sont des relations de dépendance entre les variables aléatoires qui forment alors une chaîne de causalité.

Un réseau bayésien est en quelque sorte un réseau causal, auquel on ajoute des éléments de probabilité issus de la théorie bayésienne. Cette dernière est basée sur la fameuse règle de Bayes qui dit que, si l'on a deux événements A et B, alors :

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}$$

La formule de Bayes est généralement exploitée en ignorant P(B). Le but recherché alors est non pas la probabilité précise de réalisation d'un événement, mais la confiance que l'on a dans la réalisation de cet événement en se fiant à l'expérience acquise.

Plusieurs travaux ont utilisé les réseaux bayésiens pour prédire la qualité de logiciel. Neil & Fenton [10,11] ont conduit des travaux pour prédire la densité des défauts et le coût de développement des bibliothèques de classe à objet. Selon la Figure1, les défauts introduits dépendent principalement de la complexité du problème et de l'effort de conception.

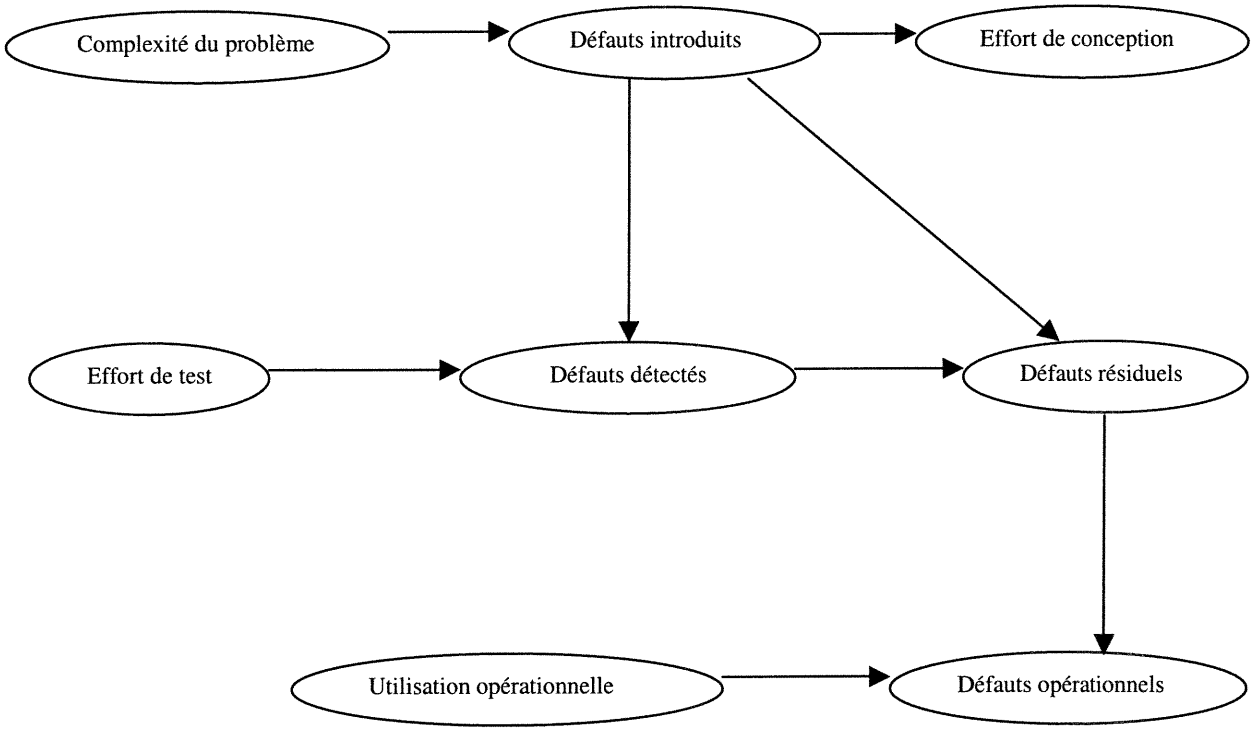


Figure 1. Estimation des défauts de développement du logiciel en utilisant un réseau bayésien.

Les résultats montrent qu'il est très probable que l'utilisation opérationnelle est susceptible d'être très grande et la complexité de problème est également grande si le nombre de tests faits est petit.

Un réseau bayésien permet de modéliser de manière explicite des situations incertaines, de combiner différents types d'informations et d'ajouter plus de clarté et de visibilité

pour la prise de décision en utilisant des outils graphiques tel que **Hugin** [12] pour les calculs et la représentation de réseau bayésien.

2.5 APPROCHES PAR ARBRE DE DÉCISION

Un arbre de décision est un graphe dont les nœuds non-terminaux sont associés à une question. Les réponses sont associées aux branches menant aux enfants d'un nœud non-terminal. Un chemin de la racine à un nœud correspond à une série de questions et réponses. Les items stockés dans un nœud terminal sont ceux qui correspondent à la série de questions et réponses qui constituent le chemin de la racine au nœud. Ici, les questions portent sur les attributs et leurs valeurs par rapport à des seuils. Les questions et réponses sur le chemin de la racine à un nœud déterminent une conjonction d'attributs définissant un sous-ensemble de l'ensemble des exemples possibles.
C4.5 [13] est un outil de classification basé sur l'approche par arbre de décision. Il utilise un ensemble d'exemples qui ont la même structure, de la forme attribut-valeur. Ces derniers représentent les exemples de classe.

L'étape clé de l'algorithme est la bonne sélection d'attributs pour obtenir des arbres de décisions ayant une grande exactitude de prédiction. Une mesure d'entropie est utilisée pour déterminer la quantité d'information qu'apporte le nœud. Soit un attribut A qui prend une valeur dans l'ensemble $\{a_i\}_{i=1,\ldots,n}$ et la distribution de probabilité $P=\{p(a_i)\}_{i=1,\ldots,n}$ où, $p(a_i)$ la probabilité que $A=a_i$, l'information donnée par cet attribut est donnée par l'entropie de Shannon :

$$H(A) = -\sum_{i=1}^{n} P(a_i)\log_2\left(P(a_i)\right)$$

Cette notion est exploitée pour ranger les attributs et pour construire des arbres de décision où, dans chaque nœud, on utilise l'attribut avec la plus grande valeur de discrimination.

La Figure suivante illustre la forme d'un arbre de décision construit par C4.5 (voir [14]) :

Exemple d'un arbre de décision pour DIT = 3, CLD = 2, NOM = 4 :

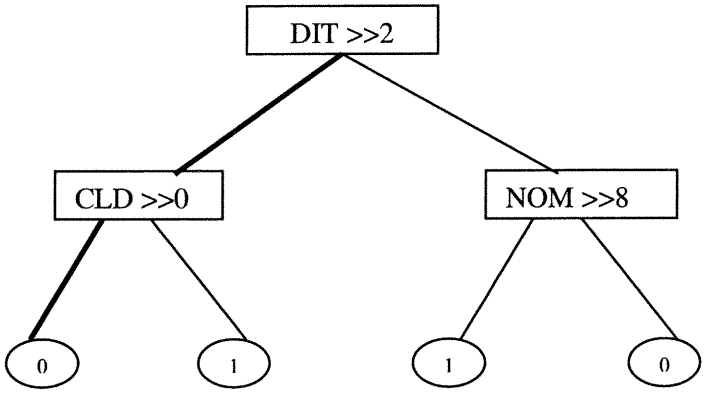


Figure 2. Classification basée sur l'inférence binaire

Dans cet exemple, l'arbre de décision est généré pour estimer la stabilité de l'interface des bibliothèques de classe. Dans les nœuds, on trouve les variables indépendantes (métriques) sélectionnées pour estimer la caractéristique s**tabilité** et, dans les feuilles, la classification obtenue (classe stable à 1, classe instable à 0). Si on considère un cas avec des valeurs définies pour les attributs, on cherche au premier nœud dans quelle mesure la condition est validée. On obtient vrai ou faux. On suit après une des deux branches de l'arbre affectées à Vrai pour la branche de gauche et faux pour la branche de droite. On continue ainsi en parcourant différentes branches de l'arbre jusqu'à atteindre l'une des feuilles.

2.6 APPROCHE PAR RÉSEAU DE NEURONES ARTIFICIELS

Conçus à l'origine par des biologistes pour modeler le fonctionnement du cerveau humain, les réseaux de neurones artificiels sont maintenant principalement utilisés dans le domaine de l'intelligence artificielle. L'idée est de modéliser l'entité de base du cerveau humain, le neurone, puis d'en assembler plusieurs afin de se rapprocher du raisonnement humain.

Les recherches dans le domaine de la prédiction de qualité de logiciel avec les réseaux de neurones ont été guidées par un ensemble d'auteurs; Gray et MacDonell fournissent des directives pour le développement des modèles prédictifs basés sur l'architecture réseaux de neurones. Cette architecture a été utilisée avec succès dans le travail présenté

par Wittig [15], qui exploite un réseau de neurones pour estimer l'effort de développement des applications orientées objet (voir Figure 3)

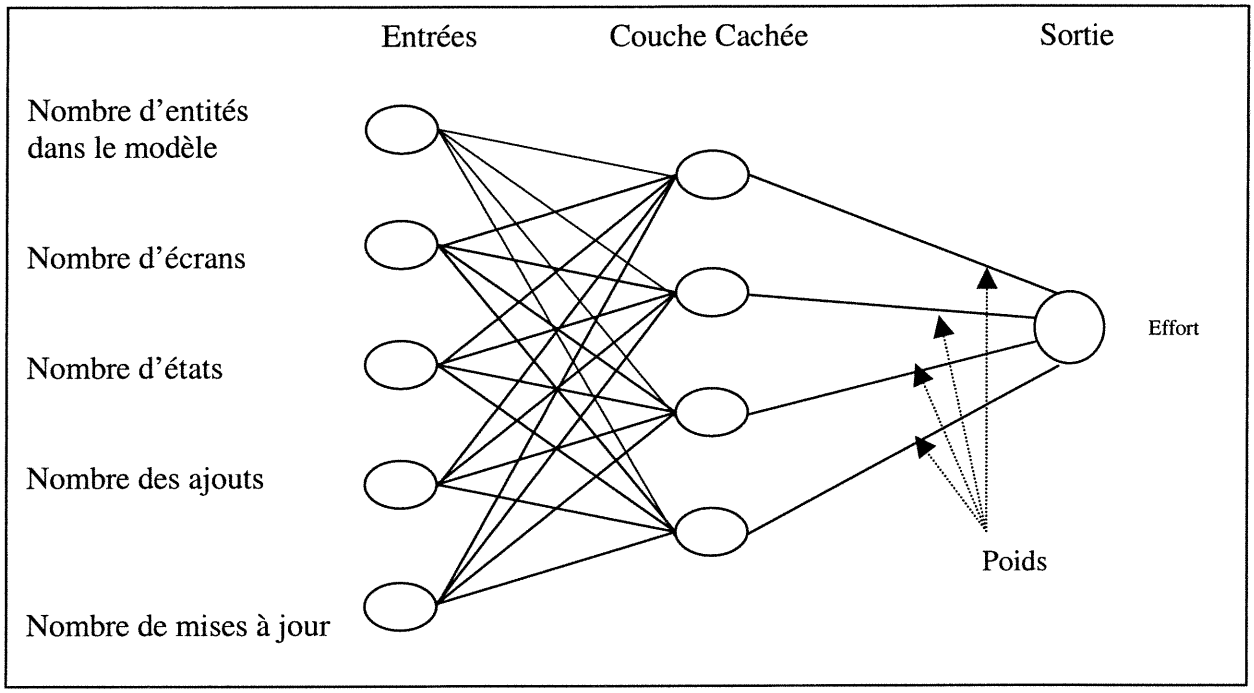


Figure 3. Exemple d'un réseau de neurone pour estimer l'effort de développement d'un logiciel.

L'exemple présenté à la Figure 3 décrit un modèle basé sur l'approche réseau de neurone, utilisant un ensemble de métriques pour prédire l'effort de développement. Le poids du réseau est déterminé, une nouvelle instance qui peut être présentée comme entrée au réseau pour estimer l'effort de développement d'un logiciel.

Les entrées du réseau représentent les mesures de qualité influençant la caractéristique de sortie qui est l'effort de développement. La couche cachée sert à représenter des connaissances en qualité.

2.7 APPROCHE PAR LOGIQUE FLOUE

Peu de publications dans le domaine de génie logiciel ont décrit l'utilisation de la logique floue dans leurs travaux [16,17], bien que les systèmes flous aient fait preuve de leur efficacité et de leur adaptation rapide au contexte des travaux en génie logiciel. Un système flou utilise une projection entre des variables d'entrée et des termes linguistiques telles que « Petit », « Moyen » et « Grand ». Munakata et Jani [18] fournissent une bonne introduction aux systèmes flous et expliquent leurs composantes principales. Il a été démontré que les systèmes flous peuvent faire l'approximation de fonctions arbitraires au même titre que les réseaux de neurones.

Les systèmes flous sont construits en trois étapes: la construction des fonctions d'appartenance qui décrivent le degré d'appartenance d'une grandeur physique à un ensemble flou; la base de règle obtenue par les experts qui servira d'évaluer pour chaque étiquette de sortie son degré de vérité par rapport aux valeurs mesurées à l'entrée; l'étape de déffuzzification qui s'explique par une combinaison des règles pour la génération des sorties.

Nous exposons ci dessous un exemple de système flou pour l'estimation de la durée de développement d'un logiciel.

Exemples :

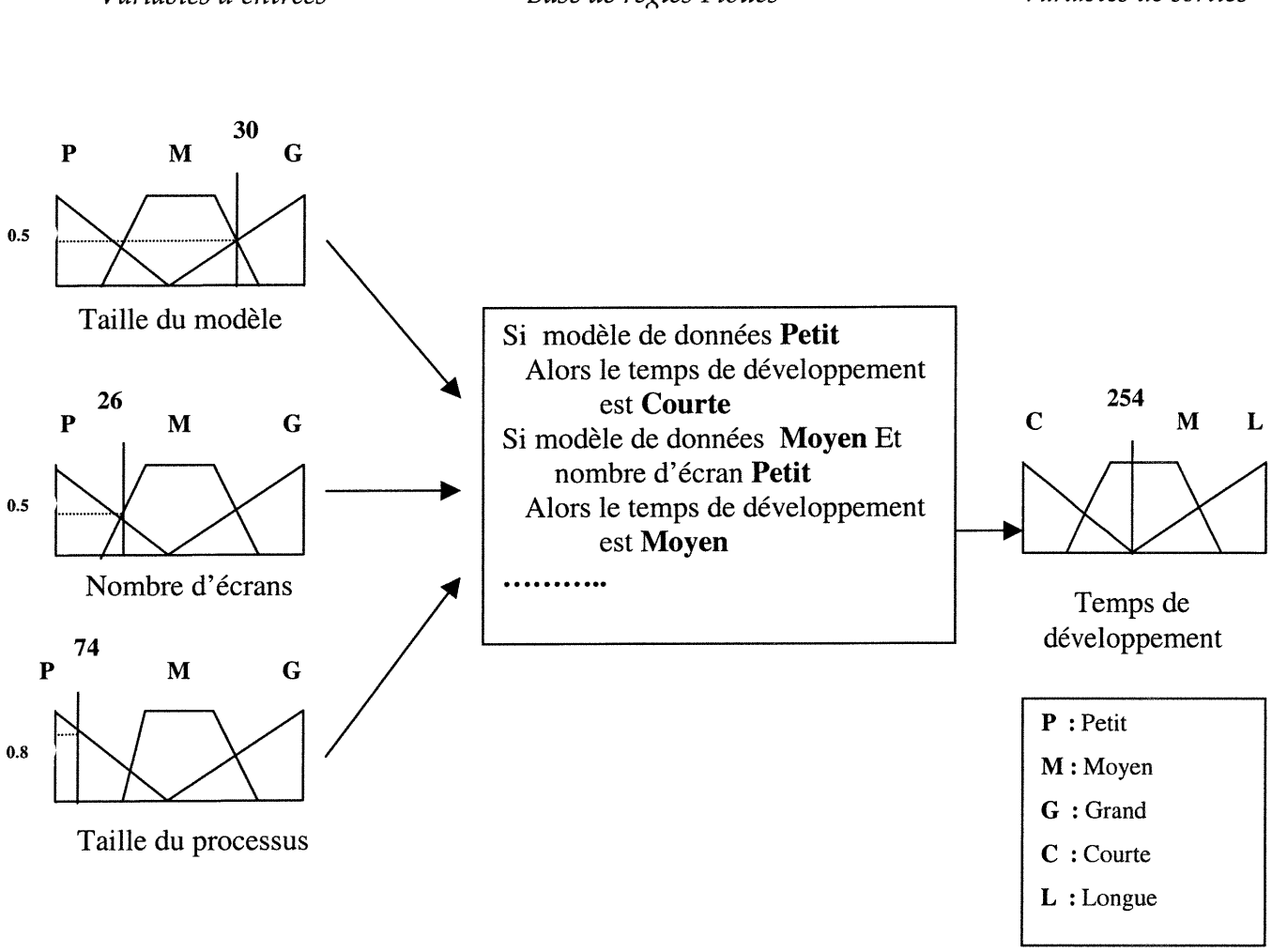


Figure 4. Système flou pour prédire la durée de développement d'un logiciel.

L'exemple de la Figure 4 décrit clairement les étapes de raisonnement d'un système flou en commençant par la fuzzification des valeurs d'entrées qui valent 30 pour la taille du modèle, 26 pour le nombre des écrans et 74 pour la taille du processus. Ces valeurs ont des valeurs de vérité obtenues à partir des fonctions d'appartenance correspondantes et qui sont respectivement 0.5, 0.5 et 0.8. Après une étape d'évaluation des règles sur les entrées choisies, une étape de défuzzification permet de combiner les valeurs de sorties pour avoir une seule valeur de sortie qui est 254. Cette valeur de sortie précise

donne le temps de développement recherché; elle est obtenue par l'une des techniques de défuzzification connues comme le centre de gravité ou la moyenne pondérée, etc.

2.8 AUTRES APPROCHES

Plusieurs travaux ont été réalisés dans le but est combiner deux approches d'intelligence artificielle, l'exemple type étant le système hybride Neuro-Flou qui est la combinaison des approches par logique floue et par réseaux de neurones [19,20]. Cette combinaison exploite les forces des deux approches et évite les défauts de chacune. Ces techniques diffèrent sur plusieurs points mais partagent les mêmes principes de bases; ce sont des systèmes adaptatifs qui peuvent traiter des règles linguistiques facilement compréhensives permettant l'initialisation d'un réseau basé sur des connaissances disponibles.

Une autre approche basée sur le raisonnement à base de cas trouve application dans le domaine du génie logiciel. Cette dernière est une méthode de stockage des observations qui peuvent prendre la forme de données de natures différentes (données sur les spécifications de projets, sur l'effort demandé pour l'implémentation, etc.) L'approche de raisonnement à base de cas est composée d'un pré-processeur pour préparer les entrées, une fonction de similarité pour rechercher les cas similaires dans une base de cas, un estimateur pour estimer la valeur de sortie, et un modificateur de mémoire pour ajouter les nouveaux cas à la base de cas s'il y a lieu [21]. Les experts, à l'aide de cette technique, peuvent faire des comparaisons sur les cas précédentes. Plusieurs travaux dans le domaine de génie logiciel ont utilisé cette technique dans leurs projets. L'exemple de travaux de Mukhopadhyay et al [22] qui comparent la performance d'un système de raisonnement à base de cas, un expert humain et un modèle utilisant les points de fonctions et COCOMO. Ces travaux ont prouvé la performance de l'approche de raisonnement par cas en comparaison avec les autres approches.

2.9 ÉTUDE COMPARATIVE DES TECHNIQUES DE PRÉDICTION

Le tour d'horizon que nous venons de faire sur les modèles de prédiction de qualité de logiciel nous a permis de développer une idée sur ces techniques; leurs principes de fonctionnement, leurs stratégies de raisonnement et leurs capacités de prédiction.

Nous jugeons très utile de présenter une étude publiée par Andrew R. Gray et Stephen G. MacDonell [23] qui compare les différents modèles de prédiction décrits plus haut en se basant sur un ensemble de critères de modélisation pour bâtir ces modèles. Le tableau comparatif suivant résume ces résultats :

Tableau 1 : Comparaison entre techniques de prédiction de point de vue de la capacité de modélisation

Techniques	Modèle disponible	Supporte d'autres techniques	Explique la sortie du modèle	Adapté aux petits ensembles de données	Peut être ajusté pour de nouvelles données	Processus de raisonnement visible	Adapté aux modèles complexes	Inclut des faits : connaissances	Application pour les métriques de logiciel
Régression des moindres carrés	***Non***	***Non***	***Partiellement***	***Non***	***Non***	***Oui***	***Non***	***Partiellement***	Des modèles simples impliquant un petit nombre de variables et des relations linéaires ou linéaires après transformation.
Régression robuste	***Non***	***Oui***	***Partiellement***	***Partiellement***	***Non***	***Oui***	***Non***	***Partiellement***	Quand les données contiennent un certain nombre d'éléments d'influence c'est un modèle général qui adapte des projets normaux.
Réseau de neurones	***Oui***	***Non***	***Non***	***Non***	***Partiellement***	***Non***	***Oui***	***Partiellement***	L'exactitude des métriques choisies est beaucoup plus importante que la compréhension des relations dans le réseau. Ce dernier peut être utilisé pour regrouper des systèmes dans des groupes semblables. Ceci peut aider à modéliser les différents systèmes, et à séparer des modèles qui doivent être développés en utilisant d'autres techniques.
Système flou	***Oui***	***Partiellement***	***Oui***	***Oui***	***Partiellement***	***Oui***	***Oui***	***Oui***	La première estimation pour des modèles plus détaillés avec suffisamment d'information n'est pas complète (évite excessivement l'engagement précis aux valeurs spécifiques et représente l'incertitude des évaluations à ce niveau). Peut également être utile lorsque les données sont seulement disponibles en petite quantité.

Systèmes à base de règles	***Non***	***N/A***	***Oui***	***N/A***	***N/A***	***Oui***	***Oui***	***Oui***	Fournit un processus piloté par des données pour développer des modèles flous permettant la mise en place de l'extraction des règles. Cela peut aider à la compréhension du processus de développement. Demande plus de données qu'un système flou, mais peut fonctionner mieux qu'un réseau de neurone.
Système hybride Neuro – Flou	***Oui***	***Partiellement***	***Oui***	***Partiellement***	***Partiellement***	***Partiellement***	***Oui***	***Oui***	Systèmes déterministes lorsque les relations reflètent peu de comportements stochastiques
Raisonnement à base de cas	***Oui***	***Partiellement***	***Oui***	***Partiellement***	***Oui***	***Partiellement***	***Oui***		Les situations pour les projets avec les mêmes variables indépendantes qui tendent à être semblables en termes de variables dépendantes, mais les relations sont très complexes. Peut servir comme support utile pour le raisonnement des experts par analogie.
Arbre de régression	***Oui***	***Oui***	***Oui***	***Partiellement***	***Oui***	***Partiellement***	***Oui***	***Partiellement***	L'arbre de régression permet à différents modèles d'être utilisés par morceaux pour différents types de système. Fournit de bonnes facilités d'explication.
Classification ou arbre de décision	***Oui***	***Oui***	***Oui***	***Partiellement***	***Oui***	***Partiellement***	***Oui***	***Partiellement***	Adapté pour les variables non numériques. Peuvent être utilisés pour produire des données théoriques aussi bien que des modèles de prédiction.

N/A : Non Applicable

Tableau 1 (Suite)

Le Tableau 1 représente une comparaison entre les techniques de prédiction en fonction d'un ensemble de critères d'évaluation. Les techniques présentées ne conviennent pas toutes à tous les types de problèmes. Cependant il y a d'autres critères qui peuvent être mis en jeu et qui peuvent influer sur les performances d'une technique de prédiction. En plus, pas tous les facteurs qui peuvent influencer la sélection de technique sont énumérés ici; d'autres peuvent inclure le logiciel de modélisation, par exemple C4.5 [13] pour la génération des arbres de décisions, **Hugin** [12] pour la représentation des réseaux bayésiens, etc.

Les colonnes du tableau représentent les éléments de modélisation pour chaque technique de prédiction : le modèle disponible réfère à l'habilité de la technique de prédiction à déterminer sa propre structure sans l'intervention des développeurs pour fournir la relation entre les entrées et les sorties. La deuxième colonne représente la robustesse de l'estimation face à des données externes. Quelques techniques sont capables de fournir une explication de leur raisonnement; cela est mentionné dans la colonne trois. Les techniques de modélisation dépendent de la représentativité de la taille des échantillons utilisés. Cependant, en incluant les connaissances du domaine, ces modèles deviennent plus précis. La colonne suivante explique l'adaptabilité des techniques à l'ajout de nouvelles données. La notion de visibilité dans le processus de raisonnement de ces techniques pour l'utilisateur est très importante. L'adaptabilité des techniques aux modèles complexes facilite l'ajout des connaissances des experts du domaine. Le tableau couvre la capacité pour chaque technique d'inclure des informations connues (connaissances). Finalement la dernière colonne du tableau décrit les applications des techniques de prédictions pour les métriques de logiciel.

2.10 SYNTHÈSE

Les techniques utilisées pour construire des modèles de prédictions de la qualité de logiciel citées plus haut donnent une vision générale sur leurs structures de fonctionnement, les étapes de leur raisonnement et leurs capacités de modélisation.

Dans l'ensemble des critères considérés dans le tableau 1, il est sûr que certaines méthodes d'analyse et de modélisation non classique offrent des possibilités intéressantes de construction des modèles prédictifs utiles et robustes. En particulier, la technique basée sur le raisonnement par cas et l'approche basée sur l'arbre de régression peuvent être favorisés en premier lieu, car elles sont simples à utiliser, pertinentes et appliquant intuitivement des méthodes de classification et d'évaluation. D'autre part, certaines techniques présentent des capacités limitées dans la construction des modèles de prédiction tels que certaines techniques statistiques. D'autres techniques fonctionnent en boîte noire: elles ne fournissent pas des explications sur leur processus de raisonnement; c'est le cas des réseaux de neurones.

Il est évident que l'utilisation de telles méthodes ne va pas fournir toutes les solutions aux problèmes d'analyse et de prédiction. Cependant il est nécessaire qu'une technique représente correctement les variables d'entrées, dérive les relations complexes liées à la construction des modèles prédictifs et aboutisse à des résultats significatifs. D'autres indicateurs comme l'interopérabilité, l'interprétabilité, la compréhensibilité, l'exactitude et la performance peuvent être des indicateurs d'importance primordiale dans un modèle de prédiction.

Enfin, la sélection d'une technique pour la construction d'un modèle prédictif nécessite la considération d'un ensemble de critères pertinents à l'étude en question.

2.11 CONCLUSION

À la lumière de ce qui a été présenté sur les techniques de construction des modèles prédictifs, nous pouvons conclure que certaines de ces techniques produisent:

- des modèles qui utilisent des valeurs seuils qui sont elles-mêmes dépendantes de l'échantillon,
- des modèles naïfs dont les variables sont relativement distantes au niveau de l'interprétation,
- des modèles dont l'applicabilité dépend de la représentativité de l'échantillon utilisé,
- des modèles ''Boite noire'' dont le processus de raisonnement est invisible.

Notre besoin est d'utiliser des techniques permettant de palier les faiblesses des anciennes techniques et de construire des modèles qui devrait tenir compte de ce qui suit:

- Les différents processus
- Les variables empiriques
- Le jugement des experts
- Les relations de cause à effet
- La notion d'incertitude
- L'information incomplète

Nous proposons dans notre choix de solution une approche causale basée sur l'utilisation de la logique floue comme technique de modélisation des entrées et sur les heuristiques du domaine afin étendre les modèles de départ. Nous insistons sur le fait que c'est la combinaison logique floue plus les heuristiques du domaine qui fera l'objet de notre approche. Nous avons exploité pour cela des métriques d'héritage et de couplage pour évaluer la stabilité de bibliothèques de code orienté objet écrites en java et réparties sur plusieurs versions.

Nous expliquons en détail cette approche en donnant une présentation sommaire sur la logique floue dans le chapitre suivant.

Principes des systèmes flous

Ce chapitre introduit les concepts de base de la logique floue. Nous apportons le détail des principales étapes de conception floue sans énoncer les aspects mathématiques de cette dernière. Ensuite, nous éclaircissons ces étapes par le schéma explicatif d'un contrôleur flou. Enfin, nous énumérons les domaines d'application où la logique floue a fait la preuve de son efficacité.

Nous nous attarderons sur le défi d'utilisation de la logique floue, pour mettre en relief les motivations qui nous ont poussées à développer notre approche.

3.1 PRINCIPE GÉNÉRAL

Le concept de la logique floue a vu le jour en 1965, grâce à Lotfi Zadeh, un professeur de génie électrique à l'université de Berkeley (USA). Il a publié un article intitulé "Fuzzy Set Theory" [24], "Théorie des ensembles floues" où il a expliqué les grands aspects de son approche basée sur une théorie mathématique des ensembles flous.

La logique floue est une technologie pour le traitement de connaissance imprécises basées sur des termes linguistiques, telle que froid, chaud, grand, etc., c'est-à-dire analogues à ceux utilisés pour la transmission des connaissances entre les personnes. Ce "Naturel" de l'énonciation et du traitement est l'une des raisons de l'application de plus en plus fréquente de la logique floue.

Un exemple : dans la logique binaire, une température peut être qualifiée par les termes décisifs, froids ou chauds. Dans la logique floue, des échelons d'appréciation intermédiaires du paramètre température sont également possibles, une température peut être, par exemple, plus ou moins élevée.

La logique floue se propose de formaliser l'usage de termes vagues d'un domaine, dans le but de les rendraient manipulables par les ordinateurs. Il deviendra alors

possible de remplacer des modèles mathématiques ou logiques par des modèles basés sur des descriptions verbales simples.

Classée parmi les techniques de l'intelligence artificielle, la logique floue permet de modéliser puis de remplacer l'expertise de conduite de projets, expertise en provenance du concepteur ou de l'utilisateur.

La logique floue gagne en popularité car il s'agit d'une approche essentiellement pragmatique, efficace et générique. On dit parfois qu'elle permet de systématiser ce qui est du domaine de l'empirisme, et donc difficile à maîtriser. La théorie des ensembles flous fournit une méthode pertinente et facilement réalisable dans des applications temps réel; elle permet de transcrire et rendre dynamiques les connaissances des concepteurs ou des opérateurs.

La logique floue ne remplace pas nécessairement les systèmes anciens. Elle est complémentaire. Ses avantages viennent de ses capacités à:

- Formaliser et simuler l'expertise d'un opérateur ou d'un concepteur dans la conduite et le réglage d'un procédé,
- Donner une réponse simple pour des propositions dont la modélisation est difficile,
- Prendre en compte sans discontinuité des cas ou exceptions de natures différentes, et les intégrer au fur et à mesure dans l'expertise,
- Donner une formulation plus simple, plus facile à comprendre, des solutions plus près des humains, des opérateurs de machine,
- Permettre un entretien plus facile et un dépannage aisé,
- Prendre en compte plusieurs variables et effectuer de la « fusion pondérée » des grandeurs d'influence,
- Permettre à la fois la représentation de l'imprécision sous forme d'ensemble flous et la quantification de l'incertitude par les nombres flous et des degrés d'appartenance,
- Etre robuste, c'est-à-dire résister bien aux perturbations qui peuvent affecter le processus.

Tous ces avantages nous ont poussés à choisir la logique floue comme technique de modélisation de notre système et de construction de notre modèle de prédiction.

3.2 ÉTAPES DE CONCEPTION FLOUE

Les démarches de conception d'un système flou suivent les étapes suivantes:

- Diviser l'univers de discours en étiquettes floues(exemple: Grand, Moyen et Petit)
- Modéliser les entrées du système par des courbes de "fonctions d'appartenance" qui donnent les degrés d'appartenance à différents états identifiés pour ces entrées.
- Établir des règles d'inférence liant les entrées aux sorties.
- Évaluer pour chaque étiquette de sortie son degré de vérité par rapport aux valeurs mesurées à l'entrée.
- Prendre pour chaque valeur de sortie la moyenne pondérée des degrés de vérité obtenus

Les composantes de la logique floue sont: les ensembles flous pour la représentation de variables linguistiques, les fonctions d'appartenance qui décrivent le degré d'appartenance de grandeurs physiques (profondeur dans l'arbre d'héritage, nombre de méthode dans une classe, couplage entre classe) à un ensemble flou (grand, petit, moyen) et les opérateurs flous qui permettent l'énonciation de relations logiques entre les assertions floues (conclusions du genre 'Si, Alors').

Nous présentons dans le schéma suivant le principe de fonctionnement d'un contrôleur flou.

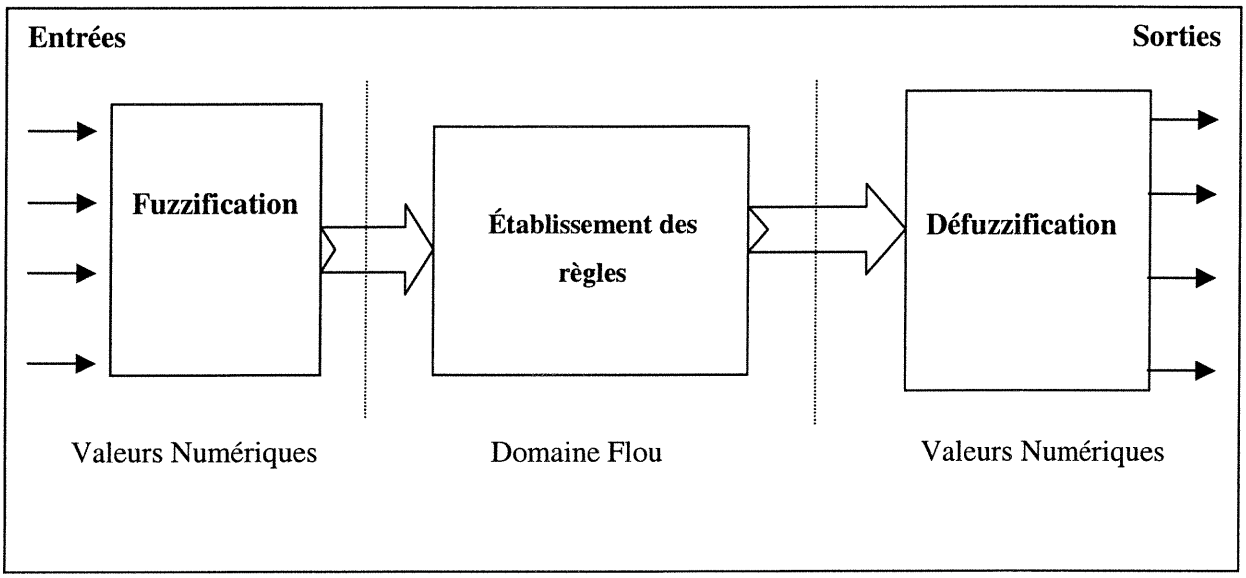


Figure. 5 Conception d'un contrôleur flou

Le schéma suivant donne une représentation plus lisible du processus de modélisation d'un système flou; celui ci est composé de : La quantification floue des entrées et sorties du système (fuzzification), l'établissement des règles liant les sorties aux entrées et, enfin, la combinaison des règles pour la génération des sorties (défuzzification).

La Fuzzification : consiste à collecter les informations susceptibles d'être traitées (les entrées), puis définir leurs conditions d'appartenance à un système.

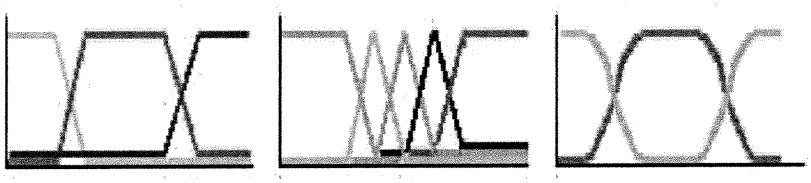
Établissement des règles (Inférence): le mécanisme d'inférence le plus couramment utilisé est celui dit de "Mamdani" . Il représente une simplification du mécanisme plus général basé sur "l'implication floue" et le "modus ponens généralisé" .

Defuzzification : détermine la façon dont seront exploitées les données ainsi collectées. Le principe consiste à leur attribuer une notation variable, comprise entre 0 et 1, afin de parvenir à une valeur précise. C'est un passage du "monde flou" au "monde réel" .

3.2.1 La fuzzification

La première étape du traitement d'un problème par la logique floue consiste donc à modéliser chacune des entrées du système par des courbes donnant les degrés d'appartenance à différents états identifiés pour ces entrées. Cette étape est appelée la fuzzification, celle ci peut s'appliquer aussi bien aux variables d'entrée que de sortie.

Les courbes d'appartenance prennent différentes formes en fonction de la nature de la grandeur à modéliser. Elles présentent une classification des valeurs d'entrées et de sorties en zones graphiques de la forme suivante :



La fuzzification des variables d'entrée est une phase délicate du processus mis en oeuvre par la logique floue. Elle est souvent réalisée de manière itérative et requiert de l'expérience.

3.2.2 Établissement des règles (Inférence)

La deuxième étape dans un processus de modélisation flou est l'établissement des règles liant les entrées aux sorties. Cette étape peut aussi être appelée l'inférence des règles. Elle consiste à construire une base de règle floues en s'appuyant sur les connaissance issue de l'expertise humaine.

Dans la pratique, les règles font appel à des conditions d'entrée plus complexes mettant en oeuvre des conditions logiques du type, "ou", "et" ou non. Il existe plusieurs lois de composition, mais la plus couramment utilisé est la suivante :

Quand les conditions sont liées par une logique "ou", on considère le degré d'appartenance **maximum** parmi les conditions d'entrée. Quand les conditions sont liées par une logique "et", on considère le degré d'appartenance **minimum** parmi les conditions d'entrée.

3.2.3 Défuzzification

La dernière étape dans le processus flou est la transformation des valeurs floues de sorties résultant de l'inférence des règles en des valeurs précises directement exploitables par l'utilisateur du modèle. C'est un passage du monde flou au monde réel qui est appelée défuzzification.

Les méthodes de défuzzification souvent utilisées sont:

- La technique du maximum
- La technique de la moyenne pondérée
- La technique du centre de gravité

La dernière est la plus performante: elle consiste à tracer, sur un même diagramme, les différentes formes correspondant au fonctions d'appartenance associées avec les différents résultats règles, et à calculer le centre de gravité de la zone consolidée. Cette méthode est, de loin la plus coûteuse en puissance de calcul, mais elle donne les meilleurs résultats.

3.3 DOMAINES D'APPLICATION DE LA LOGIQUE FLOUE

La logique floue est une approche qui permet de régler de nombreux problèmes où les autres techniques peinent par impossibilité ou difficulté de modélisation, cette dernière apporte une étonnante efficacité surtout pour:

- Les systèmes complexes dans lesquels la modélisation est difficile, voire impossible,
- Les systèmes contrôlés par des experts humains,
- Les systèmes ayant de nombreuses entrées/sorties continues ou discontinues et des réponses non linéaires,
- Quand l'observation humaine est à l'origine d'entrées ou de règles de contrôle du système,
- Dans tous les domaines où un "flou" persiste, notamment dans le domaine de l'économie, des sciences naturelles et des sciences humaines.

Au delà de l'intérêt médiatique suscité par la logique floue et de la mise en évidence d'applications commerciales ces dernières années. La logique floue permet de résoudre des problèmes par la mise en place rapide des techniques éprouvées.

Aujourd'hui, la logique floue est arrivée à sa maturité et est utilisée dans de nombreux produits "grand public" notamment au Japon. Les applications de la logique floue sont maintenant très nombreuses, dans des domaines variés et il est impossible d'en établir une liste exhaustive. La plupart a donné lieu à des réalisations industrielles. Certaines sont simples d'autres sont complexes, comme celles qui concernent la conduite d'un hélicoptère sans pilote au Japon, le contrôle de certaines usines ou une partie du guidage de la navette spatiale américaine par la NASA. La logique floue a aussi fait ses preuves dans d'autres domaines d'applications tels que la gestion de projet, le traitement d'image, les langages de programmation, les bases de données, etc..

Des modèles naïfs vers des modèles causals

Dans ce chapitre, nous présentons nos objectifs de développement d'une approche de construction d'un modèle prédictif. Nous expliquons ainsi les motivations qui nous ont amenés à choisir cette approche. Nous exposons aussi les démarches à suivre pour réaliser notre but et nous définirons la caractéristique de la qualité à étudier à savoir la stabilité. Nous faisons le choix des métriques utilisées pour estimer cette caractéristique et enfin, nous décrivons un exemple d'un modèle de prédiction classique.

4.1 OBJECTIFS ET MOTIVATIONS

Construire des modèles efficaces et utilisables pour évaluer la qualité d'un logiciel est l'un des défis relevés par les communautés de recherches et d'industrie. Les recherches dans le domaine de la construction des modèles de prédiction sont guidées par deux tendances :

La première porte sur l'utilisation des données historiquement mesurées (voir [6]). La qualité de ces derniers dépend directement de la qualité et de la représentativité des échantillons utilisés. La plupart de ces modèles capturent une tendance mais ils n'arrivent pas à expliquer l'application des valeurs seuils pour les règles qui sont générées. Par ailleurs, la majorité des modèles construits sont naïfs comme indiqué par Fenton et Neil dans [11] et n'aident pas à prendre des décisions durant le cycle de développement du logiciel; la distance cognitive entre les mesures internes (métriques) et la caractéristique de qualité à prédire est très grande.

La deuxième tendance introduit l'utilisation des connaissances extraites à partir des heuristiques du domaine pour construire des modèles prédictifs: Ces modèles invoquent des jugements d'experts pour expliquer les rapports causaux entre les variables; ce qui permet de réduire l'incertitude. Cependant ils présentent certaines faiblesses liées aux problèmes des jugements d'experts.

La plupart des modèles d'estimations construits pour prédire des caractéristiques de la qualité sont des modèles naïfs.

Les caractéristiques d'un modèle naïf sont:

- Les relations de causalité entre les variables ne sont pas bien expliquées d'où une notion d'incertitude caractérisant ces modèles.
- Les variables sont largement distantes.
- Le problème des valeurs seuils.
- Modèles dépendant de la représentativité des échantillons utilisés.

Notre but principal est de trouver comment nous pouvons rapprocher les deux tendances. Nous proposons à cet effet une approche hybride basée sur un modèle combinant les deux aspects: les échantillons de données et les heuristiques du domaine. L'idée derrière cette approche est de palier aux carences des anciens modèles et de répondre aux questions non traitées par les modèles naïfs en tenant compte des concepts oubliés par les approches classiques et qui sont:

- Capturer une tendance plutôt que les valeurs seuils.
- Expliquer les relations entre les facteurs d'influences (métriques) et la caractéristique de la qualité.
- Passer d'un simple niveau de décision à plusieurs niveaux de décision aidant à prendre des décisions intelligentes durant le cycle de vie d'un logiciel.
- Inclure des connaissances du domaine pour expliquer les relations de causalité entre les variables.
- Rétrécir la distance entre les variables d'entrée et de sortie en incluant des causes intermédiaires, par conséquent diminuer l'incertitude.
- Résoudre le problème des valeurs seuils en les remplaçant par des valeurs floues (identification des tendances en incluant d'autres causes).

4.2 DÉMARCHE À SUIVRE: DESCRIPTION DE L'APPROCHE

Comme précédemment mentionné, les modèles naïfs, construits en se basant sur des techniques statistiques ou d'intelligence artificielle, capturent efficacement les relations (tendances) entre les attributs internes et la caractéristique de la qualité de

logiciel. Cependant, ils présentent deux principaux inconvénient: premièrement, les valeurs seuils qui définissent des bornes pour les classes obtenues sont trop liées aux échantillons de données utilisés. Deuxièmement les modèles obtenus sont difficilement généralisables car il est difficile de choisir un exemple représentatif des données; cela est dû au manque de données fiables.

Il est accepté par la pratique que le problème de généralisation vient surtout des valeurs seuils plutôt que des tendances dérivées. Le deuxième problème est lié au fait que les modèles ne peuvent être utilisés pour prendre des décisions complètes durant le cycle de vie d'un logiciel.

Notre approche par modèle causal permet de résoudre les problèmes des modèles naïfs déjà mentionnés dans la section précédente et de répondre aux questions non traitées par les modèles classiques. Le nouveau modèle a comme objectifs:

- L'utilisation des valeurs floues pour capturer des tendances au lieu des valeurs seuils.

- L'ajout des heuristiques de domaine afin d'expliquer les relations de causalité entre les variables d'entrées et de sorties de notre modèle

Pour atteindre nos objectifs de construction de modèle causal nous avons opté pour:

- utiliser la logique floue pour transformer les valeurs seuils en des valeurs seuils floues.

- inclure les connaissances du domaine pour expliquer les relations entre les variables dépendantes et indépendantes et ainsi réduire la distance cognitive entre les variables qui peuvent être grande.

Intégrer des heuristiques du domaine flou à des modèles de prédiction flous nécessite une compatibilité sémantique entre les deux, ce qui n'est pas toujours le cas. Deux solutions sont envisageables pour résoudre ce problème:

1. Trouver une forme de liaison entre les heuristiques du domaine et les règles classiques.

2. Fuzzifier le modèle de prédiction avant l'ajout de connaissances du domaine.

La première solution est difficile à implémenter et à automatiser du fait que les heuristiques peuvent prendre différentes formes et que la plupart d'entre elles ont un contexte libre. Dans notre approche nous adoptons la deuxième solution qui consiste à transformer le modèle naïf à un modèle naïf flou; ce dernier est alors étendu par ajout des heuristiques du domaine.

Comme présenté dans la Figure 5, l'approche de transformation d'un modèle naïf à un modèle causal est complétée en trois étapes: les deux premières étapes transforment le modèle naïf classique en modèle flou. La troisième étape consiste à intégrer les heuristiques du domaine dans le modèle naïf flou. L'utilisation de la logique floue dans notre approche consiste à transformer les valeurs seuils des règles du modèle naïf en valeurs seuils floues. Ces étapes sont expliquées dans la Figure 5.

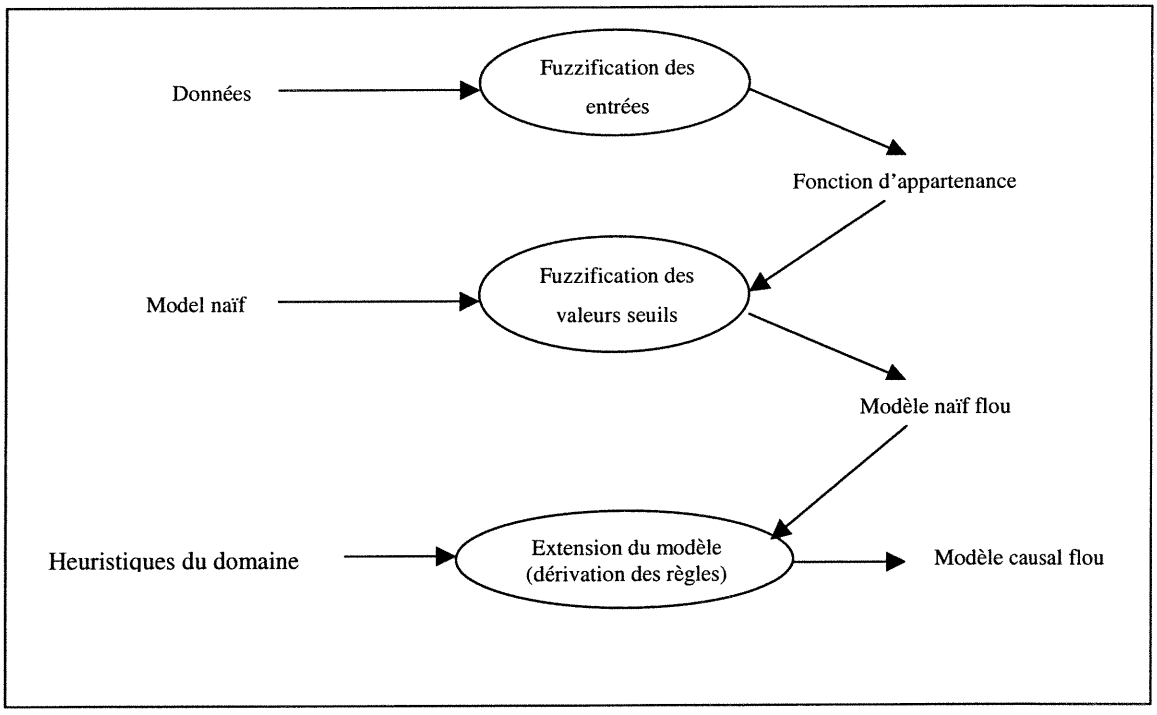


Figure 6. Approche de transformation du modèle naïf au modèle causal.

Description de l'approche

Le modèle présenté plus haut est constitué de plusieurs entités :

Entrées :

- Les données (données des échantillons),
- Un modèle naïf,
- Des heuristiques du domaine.

Traitements :

- Fuzzification des entrées,
- Fuzzification des valeurs seuils,
- Extension du modèle (dérivation de règles floues).

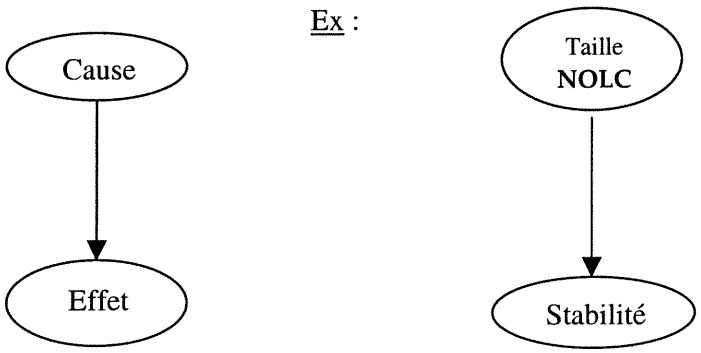
Sorties :

- Fonctions d'appartenance,
- Modèle naïf flou,
- Modèle causal.

• Les données : elles représentent des données des échantillons des bibliothèques de code Java analysées à l'aide de **DISCOVER,** un environnement d'analyse de code et d'extraction de métriques. DISCOVER englobe les fonctionnalités suivantes:

- Organiser le code sous forme d'entités,
- Générer des représentations graphiques plus abstraites du code analysé,
- Fournir une navigation facile,
- Fournir des outils pour optimiser le logiciel,
- Répondre à des requêtes formulant des critères de qualités.

• Modèle naïf : Il s'agit d'un modèle basé sur des relations de corrélation sous la forme suivante :



NOLC : Nombre de lignes de code

La relation entre la cause et l'effet est une relation de dépendance étroite. Les deux éléments sont éloignés l'un de l'autre de telle sorte qu'on ne peut pas prendre des décisions. C'est un modèle qu'on peut utiliser pour diagnostiquer et non pour décider.

• Heuristique du domaine: ce sont des règles collectées à partir de l'analyse du domaine. La classification de ces règles présente une identification d'une situation et pourquoi elle est bonne. L'identification de la situation se fait par rapport à un point de vue. Par exemple: taille (une classe d'une grande taille est stable: pourquoi ?). C'est à dire un attribut mesurable versus un autre non mesurable. Pour notre recherche nous allons nous intéresser à des règles sur les classes (méthode, variable, etc..).

• Fuzzification des entrées par l'utilisation de la logique floue comme technique de modélisation des entrées (données, règles): transformer les entrées en des labels flous en associant à chaque entrée un label.

• Fuzzification des valeurs seuils: consiste à transformer les conditions des règles (ex : DIT > 1) en des étiquettes floues (DIT est petite ou moyenne).

• Dérivations des règles floues (extensions du modèle): faite à partir de la fuzzification des entrées et des règles heuristiques du domaine. Les règles floues dérivées sont des règles liées (des règles qui aboutissent à des conclusions intermédiaires).

• Modèle naïf flou: ce modèle est obtenu en transformant le modèle naïf après fuzzification des valeurs seuils (voir la définition dans la page 9).

• Modèle causal: obtenu par l'éclatement de l'ancien modèle (modèle naïf) pour obtenir des conclusions plus rapprochées (voir la définition dans la page 9). Cette approche permet de réduire l'incertitude en introduisant des états intermédiaires. On obtient un modèle multi-critères qui aide à la décision.

L'approche que nous venons de décrire permet d'une part de pallier les faiblesses du modèle naïf en fournissant des relations causales par enrichissement des règles avec des heuristiques du domaine, d'autre part de rendre mature les relations de causalité

par l'utilisation potentielle des règles heuristique et l'utilisation de la logique floue pour fuzzifier les entrées.

Le Modèle multi-critères permet de prendre des décisions intelligentes pour réduire le risque et identifier les facteurs qu'on pourrait contrôler ou influencer. Les métriques logicielles ont été dominées par les modèles statistiques comme les modèles de régressions alors ce qui est demandé sont les modèles causals [voir article 11].

4.3 APPLICATION DE L'APPROCHE À LA STABILITÉ

Le choix de la caractéristique à étudier s'est porté sur la stabilité. Cette caractéristique est intéressante du fait qu'on peut l'extraire facilement pour disposer d'un grand nombre de données.

Le terme stabilité telle que nous l'avons envisagée se fonde sur le caractère orienté objet de notre étude: on définit la stabilité comme la stabilité de l'interface d'une classe. On s'intéresse donc à la disparition d'attributs ou de méthodes, celles-ci étant définies par leurs signatures (nom + type de retour + paramètres).

Les classifications utilisées de la stabilité sont les suivantes:

Niveau de l'impact sur la stabilité	Changement effectué sur l'interface
1 (Instable)	L'interface (Attributs, Méthodes) publique change.
2 (Stable)	L'interface (Attributs, Méthodes) publique ne change pas.

Tableau 2. Tableau de classification de la caractéristique stabilité

À travers notre étude, nous répondrions à la question:

Comment une classe évolue-t-elle d'une version d'un système à l'autre ?

Ainsi, nous vérifierons l'hypothèse suivante:

"*Le couplage d'une classe influence sa facilité d'évolution donc sa stabilité* "

4.4 EXTRACTION DES MÉTRIQUES DE LA STABILITÉ

Pour extraire les métriques qui serviront de base pour notre étude, Nous avons mis l'accent sur les métriques d'héritage et de couplage comme métriques influençant la caractéristique à étudier, à savoir **la stabilité.** Nous avons utilisé pour cela l'environnement d'analyse du code et d'évaluation de qualité de logiciel **DISCOVER** pour calculer ces métriques sur un ensemble de bibliothèque de code Java à plusieurs versions**.** Le but de ce projet est de construire un modèle de prédiction de la stabilité des bibliothèques de classes à objet à l'aide des techniques d'apprentissage avec l'outil C4.5 [25,26]. Notre intérêt porte sur les métriques orientées objets usuellement utilisés dans la littérature pour les langages orientés objets. Ces métriques peuvent être classées en métriques d'héritage, de complexité, de cohésion et de couplage.

Les métriques de stabilité choisies sont celles mesurant le couplage au sens large incluant le couplage d'héritage. Les métriques d'héritage sont celles qui sont liées à l'architecture du modèle objets, c'est à dire celles relatives à l'héritage, aux méthodes et aux attributs. Les métriques de couplage mesurent le degré d'interdépendance entre deux entités dans le développement logiciel, on parle de couplage entre modules, classes ou méthodes.

Afin d'extraire les métriques de stabilité nous avons besoin d'un ensemble de bibliothèques de code Java. Le choix de ces bibliothèques est lié à la disponibilité de versions multiples et au point de vue choisi de la stabilité: celle-ci est, en effet, graduée par le degré des modifications subies par l'interface. On peut donc la classer suivant le type d'utilisateur qui est touché par une éventuelle modification : concepteur de la bibliothèque, utilisateur des objets tels quels ou par héritage.

Grâce à l'environnement **DISCOVER** on a pu analyser les bibliothèques Java et en extraire un ensemble restreint de métriques d'héritage et de couplage. Le calcul de ces métriques est réalisé à l'aide d'un langage de script propre à Discover appelé "Access". Le tableau suivant décrit ces métriques :

Métriques de classe :

Métriques	Description	
DIT	Depth of Inheritance tree	Profondeur dans l'arbre d'héritage
NOC	Number Of Children	Nombre de sous classes directes
NPA	Number Of Public Attributes	Le nombre d'attributs publics dans une classe
NAA	Number of Available Attributes	La somme des attributs définis dans une classe et des attributs hérités
DAM	Data Access Metric	Le ratio des attributs privés et protégés au nombre total d'attributs dans une classe.
CIS	Class Interface Size	Le nombre de méthodes publiques dans une classe
NAM	Number of Available Method	La somme de méthodes définies dans une classe et les méthodes qu'elle hérite.
OAM	Operation Access Metric	Le ratio de méthodes publiques au le nombre total de méthodes déclarées dans une classe.
NPM	Average of the number of parameters per method in a class	La moyenne des paramètres par méthodes d'une classe. La somme des paramètres de toutes les méthodes par le nombre de méthodes dans une classe
NOP	Number of Polymorphic Methods	Le nombre de méthodes polymorphiques
NPT	Number of unique Parameter Types	Le nombre de types de paramètres utilisés dans les méthodes d'une classe.
OCAIC	Other Class Attribute Import Coupling	Calcule le nombre d'occurrence de relation de type CM. **CM**: Il existe une interaction Classe-Méthode entre classe **C** et **D**, si classe **C** a un paramètre de type classe **D**.
OCMIC	Other Class Method Import Coupling	Calcule le nombre d'occurrence de relation de type CA. **CA**: Il existe une interaction Classe-Attribut entre classe **C** et **D**, si classe **C** a un attribut de type classe **D**.
OCAEC	Other Class Attribute Export Coupling	Calcule le nombre d'occurrence de relation de type CI . **IC:** Import Coupling, le nombre de classes que la classe **C** utilise.
OCMEC	Other Class Method Import Coupling	Calcule le nombre d'occurrence de relation de type EC . **EC**: Export Coupling, le nombre de classe qui utilise la classe **D**.

Tableau 3. Liste de métriques de la stabilité

4.5 MODÈLE NAÏF DE LA STABILITÉ

Les modèles naïfs sont obtenus à l'aide d'une des techniques déjà mentionnées au chapitre 2 (techniques statistiques, techniques d'apprentissage, etc.). Nous avons utilisé des modèles construits par une technique d'apprentissage automatique, à l'aide de l'outil C4.5 [25,26]. Les données qui ont été servis pour l'apprentissage sont des données de métriques liées à la stabilité, ces dernières ont été extraites par l'outil d'analyse de code **DISCOVER**. Les modèles résultants prennent la forme de règles classiques qui s'écrivent comme suit :

Exemple d'un modèle naïf :

Règle 1 :

DIT > 1

-> Classe 1 [82.7%] (Instable)

Règle 2 :

DAM <= 0.7

-> Classe 1 [74.2%] (Instable)

Règle 3 :

OCAEC > 0

-> Classe 1 [70.8%] (Instable)

La règle 1 peut être lue comme suit:

> *Si la profondeur d'une classe dans l'arbre d'héritage est supérieure à 1, alors la classe correspondante est classifiée 1 avec une probabilité de 82.7%.*

Les règles obtenues possèdent comme entrées (conditions) les métriques de stabilité et aboutissent à des sorties (conclusions) qui sont une estimation de la stabilité : Classification ***Stable*** ou ***Instable***.

Chapitre 5

Mise en œuvre de l'approche d'extension d'un modèle naïf

Nous présentons dans ce chapitre les détails de l'approche de développement d'un modèle prédictif. Nous commençons en première section par la fuzzification des entrées comme première étape de modélisation de l'approche. Ensuite, nous nous attardons, dans une deuxième section, sur la description de la phase de dérivation des règles floues. Pour cela, nous définissons les notations choisies et nous construisons un modèle causal flou. Dans la troisième section, nous conduisons nos études vers l'étape de l'inférence des règles (évaluation de notre modèle). Finalement, nous justifions l'absence de l'étape de défuzzification dans notre processus de construction des modèles prédictifs de qualité de logiciel.

5.1 FUZZIFICATION DES ENTRÉES

Nous expliquons dans cette section les étapes de fuzzification des entrées. Nous adoptons la technique des histogrammes pour représenter les données de métriques et nous utilisons l'environnement Matlab pour définir les zones d'intersection des courbes obtenues. Enfin, nous construisons les fonctions d'appartenance nécessaires pour l'opération de fuzzification.

5.1.1 Technique des histogrammes

Les entrées de notre système sont des valeurs de métriques extraites à partir d'un ensemble de bibliothèques commerciales java comprenant plusieurs versions (open source). Le tableau suivant donne une description des principales applications utilisées:

Nom des applications	Nombre de Version	Nombre de Package	Nombre de fichier Total	Description	Adresse Internet
Free	6	23	182	Système pour générer des Parseurs LALR	http://www.thecouch.org/free/dl_inter.html
Jedit	9	78	1664	Vote Électronique	http://sourceforge.net/projects/jedit/
CupJava	11	34	423	Plat-forme de Serveur Web	http://www.cs.princeton.edu/~appel/modern/java/CUP/
Jetty	10	60	1089	Serveur de Servlet http	http://jetty.mortbay.com/
Jext	11	83	1059	Éditeur de texte	http://sourceforge.net/projects/jext/
Jlex	10	10	10	Analyseur Lexical	http://www.cs.princeton.edu/~appel/modern/java/JLex/
Jigsaw	3	211	2086	Éditeur de texte	http://www.w3.org/Jigsaw/

Tableau 4. Description des applications Java choisies

Plusieurs techniques de fuzzification ont été présentées dans la littérature pour fuzzifier des attributs quantitatifs. Certains d'entre eux sont simples à utiliser, d'autres sont plus sophistiqués [27,28,29]. Dans notre approche, nous utilisons une technique simple qui est basée sur la distribution des données mesurées: nous calculons pour chaque valeur d'entrée sa fréquence d'apparition dans l'échantillon et nous dérivons des partitions à partir des courbes résultantes. Les partitions représentent les étiquettes floues des métriques d'entrées, Pour certaines métriques, nous devons d'abord appliquer des pré-traitements supplémentaires sur les données d'entrées avant de dériver les clusters de la courbe de fréquence à fin de ressortir les partitions. Ces traitements peuvent être, par exemple, l'application des techniques de filtrages permettant d'éliminer les pics sortants dans une courbe. Ainsi, l'algorithme pour la fuzzification d'entrée est comme suit [30,31,32]:

- Pré-traitement sur les données d'entrée pour faire apparaître les clusters si nécessaire,
- Dessiner l'histogramme de fréquence des données d'entrées,
- Identification du nombre de groupes et assignation d'étiquettes floues à ces groupes,
- Définition des bornes et des fonctions d'appartenances associées avec un algorithme de groupement flou approprié.

Pour illustrer cette technique nous considérons le tableau suivant contenant des valeurs de métriques pour un ensemble de classes.

	DIT	NOC	NPA	NAA	.....	OCAIC	**OCAEC**
Classe 1	1	1	3	5		2	**1**
Classe 2	2	3	4	10		1	**0**
.....							
Classe n	1	2	3	8		1	**1**

Tableau 5. La distribution des valeurs de métrique OCAEC

En utilisant l'outil SPSS nous exploitons les données de métriques pour générer:

- le tableau ci-dessous qui montre les rapports entre les valeurs de métriques et leur distribution dans l'échantillon.
- l'histogramme correspondant qui est présenté dans la Figure 6.

OCAEC	Fréquences	Pourcentage
.00	1837	75.5
1.00	337	13.8
2.00	89	3.7
3.00	44	1.8
4.00	40	1.6
5.00	26	1.1
6.00	12	.5
7.00	9	.4
8.00	9	.4
9.00	5	.2
10.00	4	.2
11.00	3	.1
12.00	1	.0
13.00	6	.2
14.00	2	.1
15.00	2	.1
16.00	1	.0
18.00	2	.1
20.00	1	.0
21.00	1	.0
23.00	1	.0
26.00	1	.0
33.00	1	.0
Total	2434	100.0

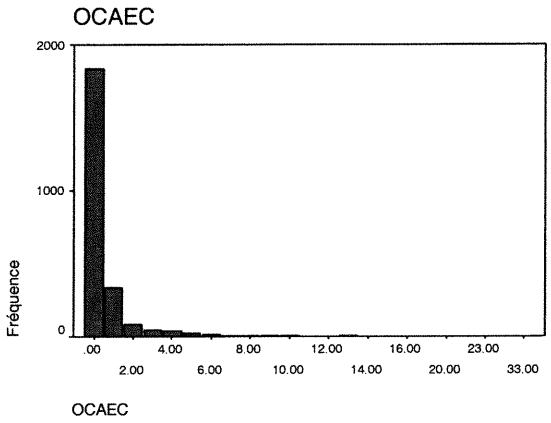


Figure 7. Histogramme des fréquences d'apparition de la métrique OCAEC

Si nous analysons la courbe obtenue (voir Figure 7), il est difficile de dériver des partitions homogènes. Du fait que la plupart des classes n'ont pas d'attributs publiques, l'échelle de l'histogramme ne donne pas une forme visuellement interprétable. Pour résoudre ce problème nous appliquons une transformation logarithmique aux valeurs de fréquences pour ressortir les petites valeurs et aplatir les plus grandes[30,31,32]. Le résultat obtenu est présenté dans la Figure 8.

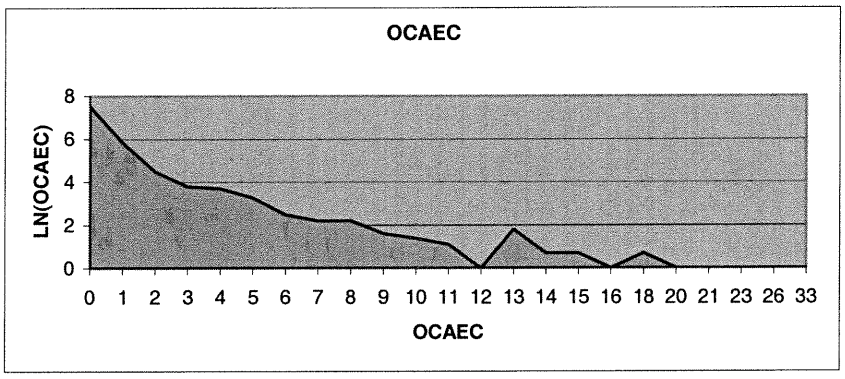


Figure 8. Courbe logarithmique de fréquences

On voit apparaître dans la nouvelle figure (Figure 8) trois partitions homogènes. À partir de ces partitions, nous pouvons définir trois labels flous pour la métrique OCAEC et associer une fonction d'appartenance à chacun. Les formes standard utilisées pour une fonction d'appartenance ont une forme trapézoïdale, triangulaire, en S ou en Z

().

N'importe quelle valeur de la métrique OCAEC peut alors être appliquée aux trois étiquettes avec différentes valeurs de vérité qui sont comprise entre 0 et 1.
Il est facile d'associer des valeurs seuils aux règles en utilisant la fonction d'appartenance obtenue. Le principe est de choisir l'étiquette qui applique le mieux la condition de la règle.

5.1.2 Utilisation de Matlab pour définir les zones d'intersections

Lors de la transformation des partitions floues en partition floues normalisées sous forme de trapèzes ou triangles, un problème à résoudre est la définition des zones d'intersections entre les groupements obtenus dans les premières étapes de la fuzzification. Nous avons utilisé l'algorithme **C_means** [29]. Cet algorithme fut implémenté à l'aide d'une fonction Matlab, FCM, qui permet de faire un regroupement flou des étiquettes choisies et de les transformer en partitions normalisées.

La fonction FCM permet de trouver des groupements flous de la façon suivante :

[CENTER, U, OBJ_FCN] = FCM (DATA, CLUSTER_N)

Cette fonction permet d'appliquer la méthode **C_means** à un ensemble de données en entrée. Les arguments d'entrée et les sorties de cette fonction sont:

DATA : l'ensemble de données à grouper; où chaque ligne représente les données d'échantillon.
CLUSTER_N : le nombre de partitions (plus grand que 1).
CENTER : les centres des partitions floues, où chaque ligne est un centre.
U : la matrice des partitions floues.
OBJ_FCN : les valeurs de la fonction objective durant les itérations.

L'exemple suivant applique la fonction **FCM** pour construire les partitions floues de la métrique OCAEC :

[C,Fm,obj]=Fcm(OCAEC,3);

Les données suivantes représentent les entrées de notre fonction : ce sont des couples valeurs de métriques – valeurs de vérités. Les valeurs de vérités données en entrée

pour la fonction sont des valeurs de vérité obtenues par une division de la colonne logarithme de la fréquence par la plus grande valeur trouvée dans cette colonne.

```
OCAEC = [0 1;1 0.77;2 0.6;3 0.5;4 0.49;5 0.43;6 0.33;7 0.29;8 0.29;9 0.21;10 0.18;11
         0.15;12 0;13 0.24;14 0.09;15 0.09;16 0;18 0.09;20 0;21 0;23 0;26 0;33 0];

[c,fm,obj] = fcm(OCAEC,3);
```

La variable C contient les valeurs de cordonnées des centres des partitions floues (étiquettes floues) :

```
C =

   24.5894    0.0031
    3.7345    0.5374
   13.1518    0.1155
```

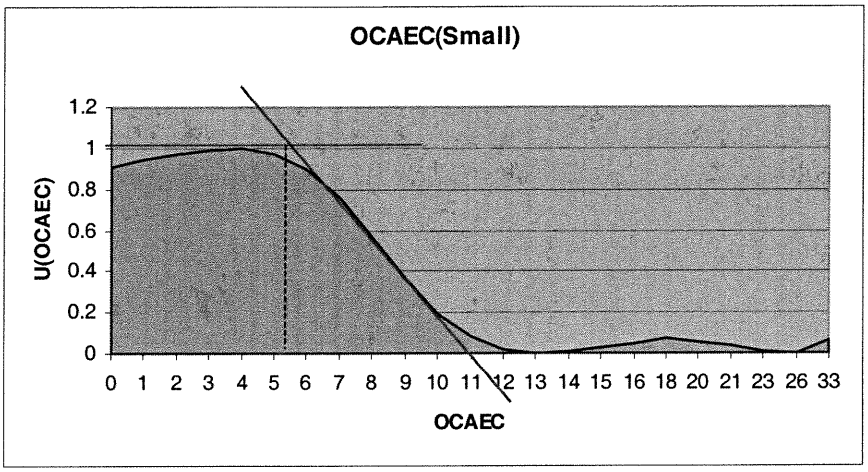
La variable **Fm** contient une matrice à trois lignes dont chacune représente les degrés de vérité de l'appartenance de chaque valeur de la métrique OCAEC à l'une de trois partitions floues. Ces données définissent les fonctions d'appartenance de la métrique OCAEC aux trois zones floues Petite, Grande et Moyenne.

Le résultat donné par Matlab est :

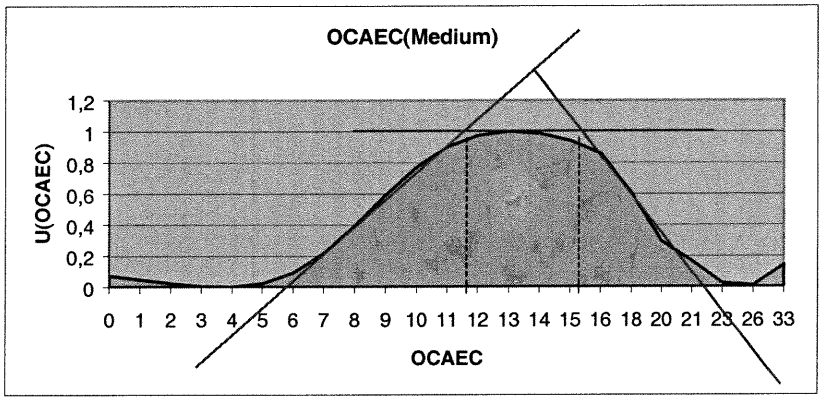
```
Fm =

  Columns 1 through 7

    0.0212    0.0127    0.0057    0.0012    0.0002    0.0041    0.0134
    0.9051    0.9395    0.9708    0.9936    0.9990    0.9723    0.8960
    0.0738    0.0478    0.0235    0.0052    0.0009    0.0236    0.0906

  Columns 8 through 14

    0.0263    0.0378    0.0420    0.0359    0.0225    0.0082    0.0003
    0.7588    0.5703    0.3665    0.1942    0.0786    0.0190    0.0004
    0.2149    0.3918    0.5915    0.7698    0.8988    0.9728    0.9993

  Columns 15 through 21

    0.0063    0.0349    0.0946    0.3267    0.6542    0.7985    0.9682
    0.0067    0.0253    0.0463    0.0697    0.0520    0.0345    0.0066
    0.9869    0.9398    0.8591    0.6036    0.2937    0.1670    0.0252

  Columns 22 through 23

    0.9842    0.7923
    0.0039    0.0654
    0.0119    0.1423
```

Donc, le résultat de l'application de la fonction **C_means,** représenté par la variable FM, donne les valeurs de vérités pour chaque partition floues. Les paramètres de cette fonction sont les données et le nombre de partitions. Nous associons à chaque donnée

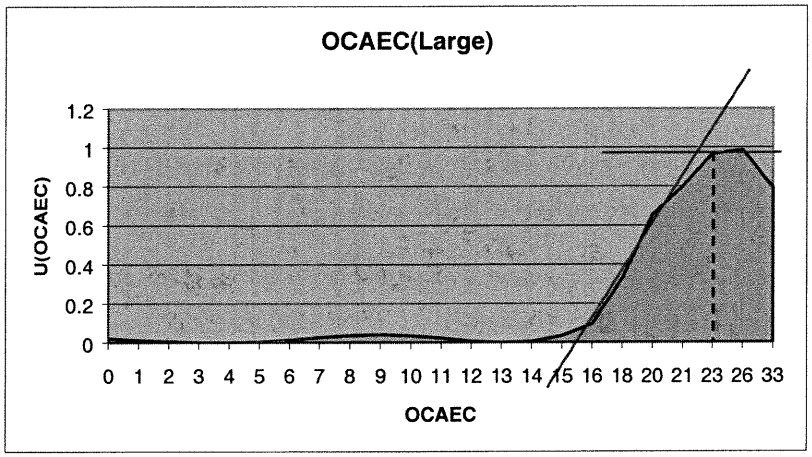
la valeur de vérité correspondante et nous traçons les fonctions d'appartenance obtenues pour chaque partition floue. Cela donne les trois fonctions d'appartenance de la métrique OCAEC suivantes:



OCAEC Petite



OCAEC Moyenne



OCAEC Grande

Figure 9. Fonctions d'appartenance de la métrique OCAEC(petite, moyenne, grande)

En analysant les fonctions d'appartenance obtenues, nous pouvons en extraire des points remarquables qui permettent de simplifier la forme de ces fonctions. Nous utilisons une technique de tangentes pour déterminer les cordonnées de ces points qui permettront de définir une des formes standard des courbes d'appartenance résultantes. Ensuite nous fusionnons les trois courbes afin de couvrir tout l'univers de la métrique OCAEC.

Les points obtenus sont :

Valeurs de OCAEC	Valeurs de Vérité
0	1
7	1
13	0
7	0
13	1
15.8	1
21.6	0
15.8	0
21.6	1
33	1

5.1.3 Génération des fonctions d'appartenance

Les résultats obtenus par l'application de la fonction **C_means** nous permettent de générer les fonctions d'appartenance pour les métriques de stabilité. Les valeurs de l'axe des ordonnées se rapportent au degré auquel la valeur de vérité d'entrée s'applique à chacune des étiquettes de la fonction d'appartenance (petit, moyen, grand). Les valeurs de l'axe des abscisses représentent les valeurs de la métrique OCAEC. Après simplification, les fonctions d'appartenance finales prennent la forme suivante :

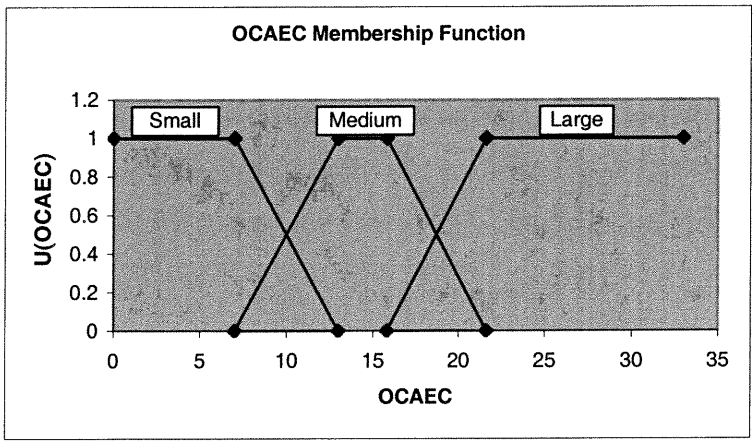
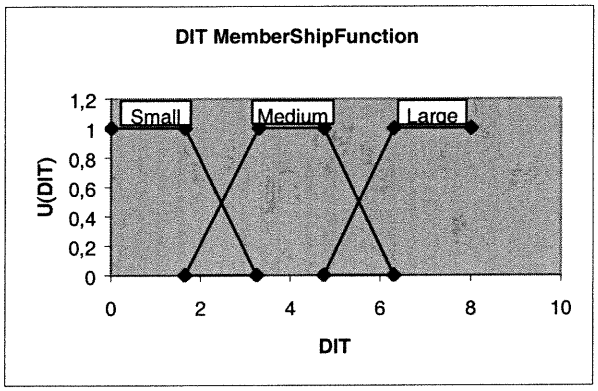


Figure 10. Fonction d'Appartenance de la métrique OCAEC

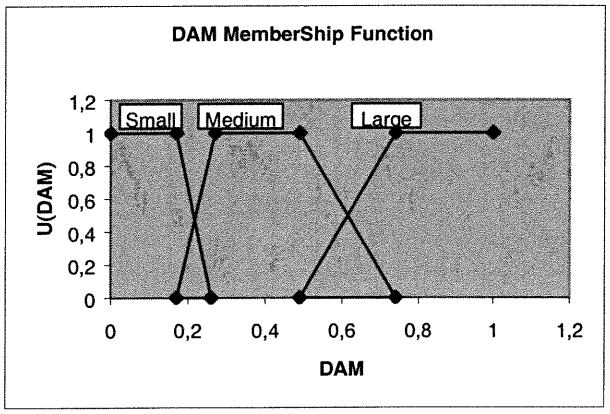
Les fonctions d'appartenance résultantes pour la métrique OCAEC décrivent trois partitions floues: petite, moyenne et grande. Chaque valeur de métrique peut appartenir à une partition où deux avec des degrés de vérité compris entre 0 et 1; ceux-là représentent le degré d'appartenance de la valeur de métrique aux différentes partitions.

Nous présentons dans le tableau ci-dessous les fonctions d'appartenance des métriques qui seront impliquées dans notre base de règles. Ces dernières sont obtenues de la même façon que précédemment pour la métrique OCAEC.

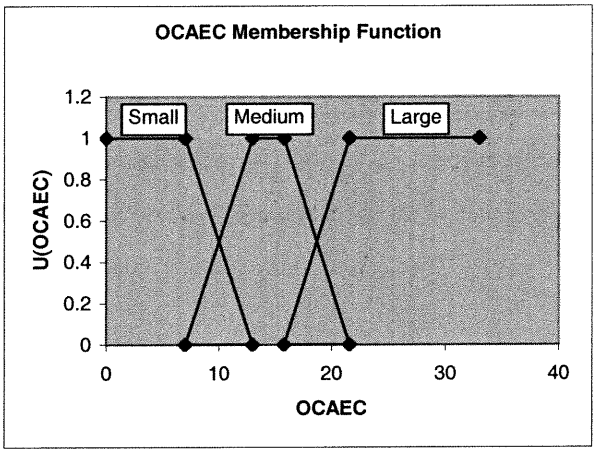
Métrique DIT



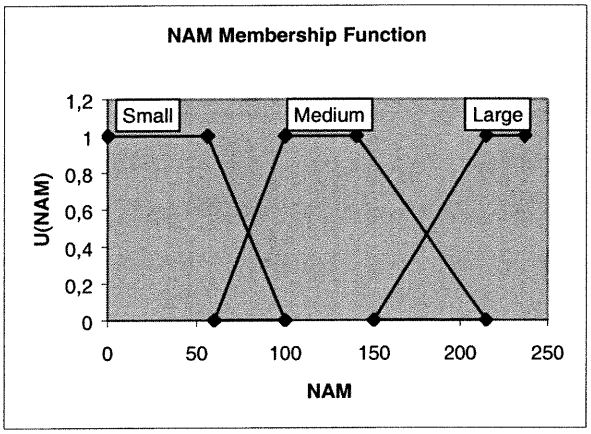
Métrique DAM



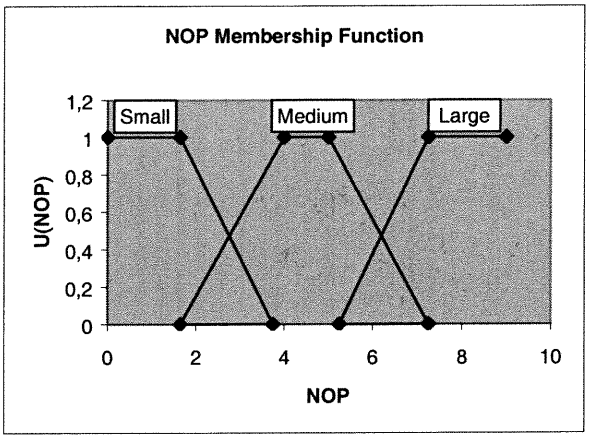
Métrique OCAEC



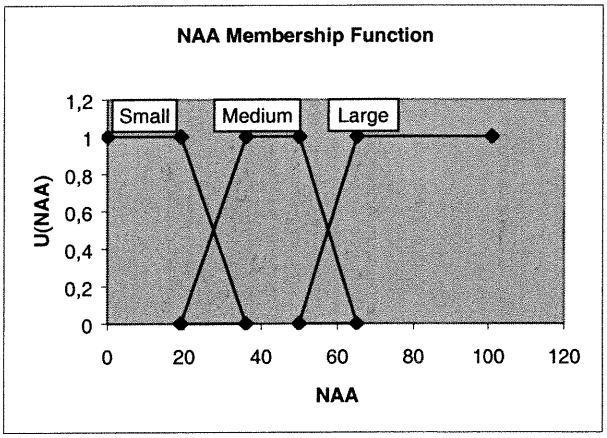
Métrique NAM



Métrique NOP



Métrique NAA



Axe des X : valeurs de métriques

Axe des Y = U(X) : valeurs de vérités

Tableau 6 : Fonctions d'appartenance pour les différentes métriques utilisées dans la base de règles.

L'étape de fuzzification sert d'entrée à l'étape de dérivation de règles floues que nous allons détailler dans les sections qui suivent.

5.2 DÉRIVATION DES RÈGLES FLOUES

Nous exposons dans cette section l'étape de dérivation de règles pour la construction d'une base de règles floues. Nous commençons par la définition du principe de dérivation de règles. Ensuite, nous décrivons l'algorithme de transformation des règles naïves en règles floues. Enfin, nous appliquons notre algorithme pour dériver les règles causales floues.

5.2.1 Principe de la dérivation

Malheureusement, bien que les règles du modèle naïf montrent des rapports raisonnables, il est difficile de les utiliser comme supports de décision. La tâche essentielle alors est d'ajouter les parties manquantes qui nous permettraient d'expliquer les relations de causalité entre les attributs internes du logiciel et la caractéristique de qualité à prédire. Pour cela, nous explosons une règle naïve en incluant des causes intermédiaires intuitivement vérifiables par un expert, en se basant sur des heuristiques du domaine [31,32]. Pour illustrer l'intégration des connaissances du domaine, prenons le cas particulier de la règle :

Si DAM(c) <= 0,7 Alors **c** peut être instable durant l'évolution du système.

DAM (Data Access Metric) est le pourcentage des attributs privés et protégés dans une classe.

La condition Si DAM(c) < = 0,7 peut être transformée en une condition floue en fuzzifiant d'abord la donnée d'entrée et en déterminant quelle(s) étiquette(s) correspond(ent) le mieux à la valeur seuil utilisée dans la condition. Dans notre cas, la condition devient Si DAM Petite ou Moyenne car la valeur seuil 0,7 correspond aux domaines de définition des étiquettes Petite et Moyenne. La nouvelle règle est visualisée dans la figure 11 où la flèche discontinue signifie que les relations entre les deux variables doivent être raffinées. Cela est accompli par la recherche des heuristiques du domaine (relations de causalité) qui peuvent connecter le fait DAM

Moyenne et/ou DAM Petite à des conclusions intermédiaires menant à la conclusion finale (classe instable).

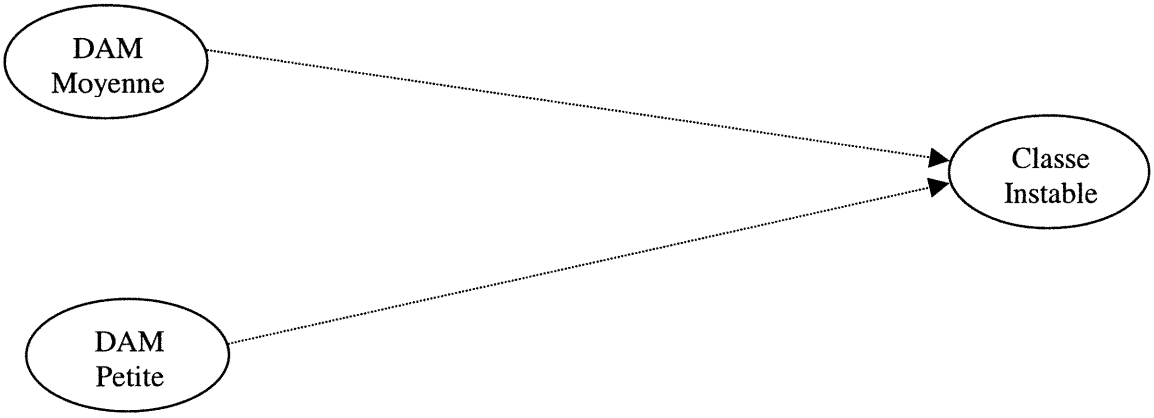


Figure 11. Règle naïve avec des valeurs seuils floue

Comme illustré dans la Figure 12, différentes relations intermédiaires peuvent être ajoutées. Parmi lesquelles « Si DAM moyenne ou petite, alors la plupart des attributs sont visibles aux autres composantes du système ». Cela peut mener à la conclusion que le couplage potentiel entre les classes étudiées et le reste du système est grand. Nous représentons cette relation avec une flèche continue pour signifier qu'elle est intuitive et vérifiable.

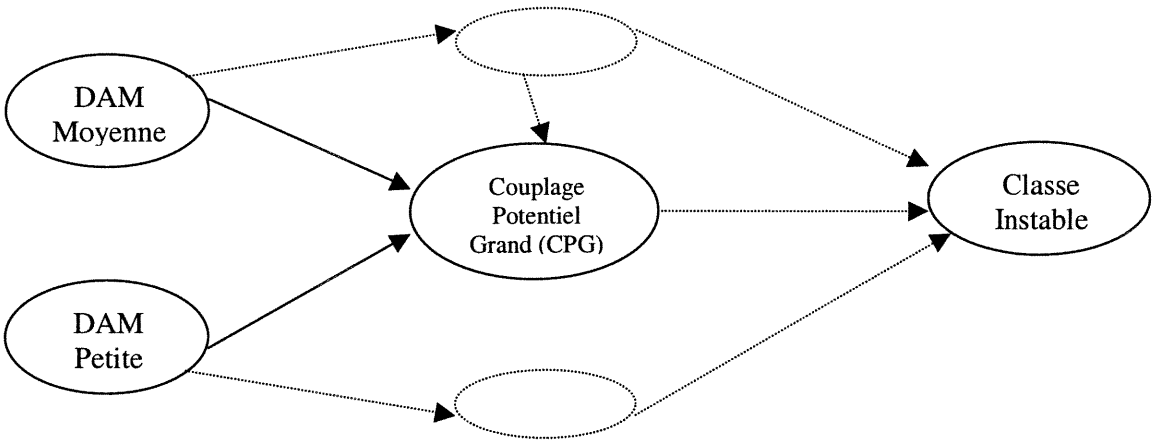


Figure 12. Introduction de premier niveau de conclusion intermédiaire

Suivant le même principe, nous devons raffiner toutes les relations représentées par des flèches discontinues. Dans notre exemple la relation entre le CPG et la classe instable peut être décomposée en utilisant d'autres heuristiques du domaine. Ainsi, un CPG peut mener à un risque d'effet de changement secondaire grand (voir Figure 14). En effet, tout changement dans un système peut produire des erreurs dans les classes étudiées. La relation entre le risque d'effet de changement (RECSG) et la classe instable est considérée comme intuitive donc elle est représentée par une flèche pleine. Le processus de raffinement continue jusqu'à ce que toutes les relations soient intuitives et vérifiables. Le modèle causal obtenu est meilleur dans son processus de raisonnement et peut alors être utilisé comme support de décision pendant le processus de développement de logiciel [30,31,32].

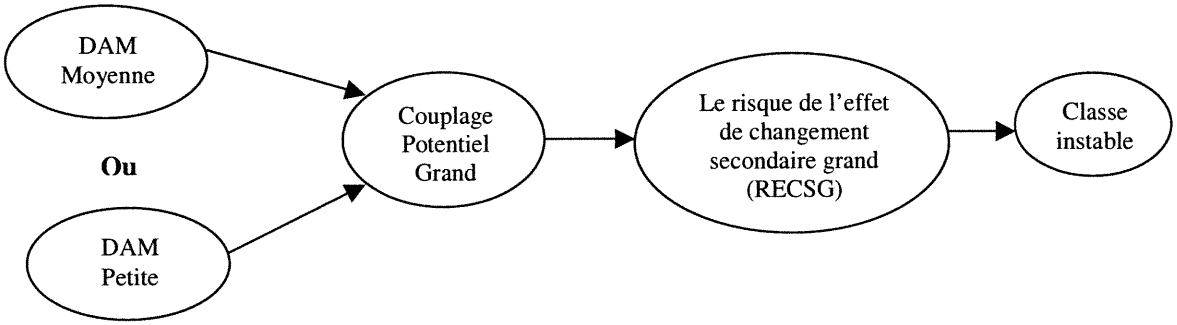


Figure 13. Raffinement final de la règle présentée dans la Figure 11

Nous raffinons dans ce qui suit la règle du modèle naïf « Si OCAEC > 0 Alors Classe instable ». Nous ajoutons une autre métrique NAA (voir la définition dans le tableau qui donne la liste des métriques de la stabilité) qui a un rôle dans la détermination de la conclusion intermédiaire portant sur l'effet de modification d'un attribut (EMAG). Nous continuons à inclure des heuristiques du domaine jusqu'à obtention de relations intuitivement vérifiables par des experts du domaine. L'éclatement du modèle naïf pour dériver des règles causales de la métrique OCAEC se traduit par le schéma suivant :

OCAEC: Other Class Attribute Export Coupling

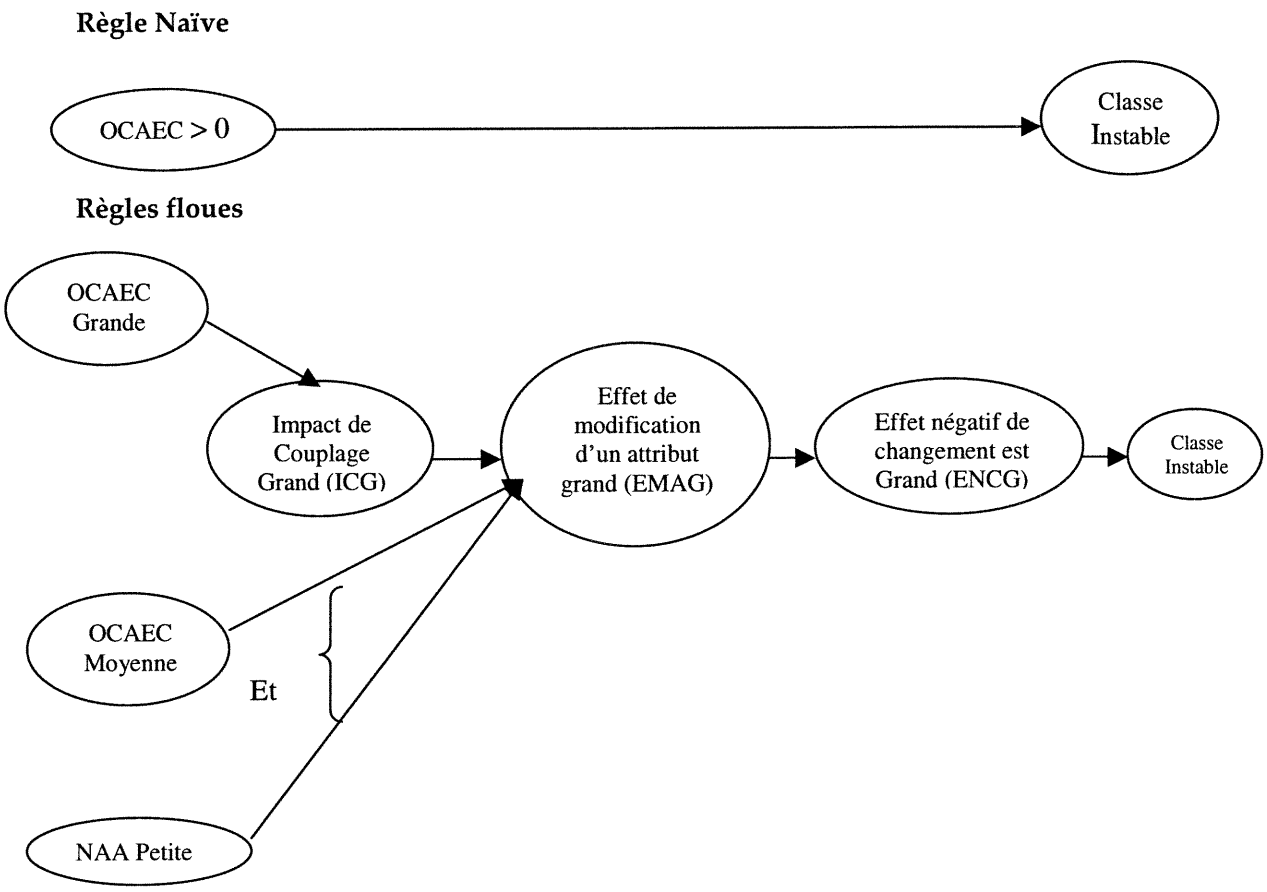


Figure 14. Dérivation des règles floues à partir d'une règle naïve et des connaissances du domaine: métrique OCAEC

Nous décomposons selon le même principe la règle du modèle naïf « Si DIT > 1 alors classe instable » en incluant des heuristiques du domaine. Dans cet exemple, nous jugeons utile d'ajouter deux nouvelles métriques qui joueront un rôle dans la détermination de la conclusion intermédiaire ou finale. Ces deux métriques sont NAM et NOP. La dérivation des règles causales de la métrique DIT se traduit par le schéma suivant [31,32]:

DIT: La longueur du chemin de la Racine de l'arbre d'héritage à la classe Règle Naïve.

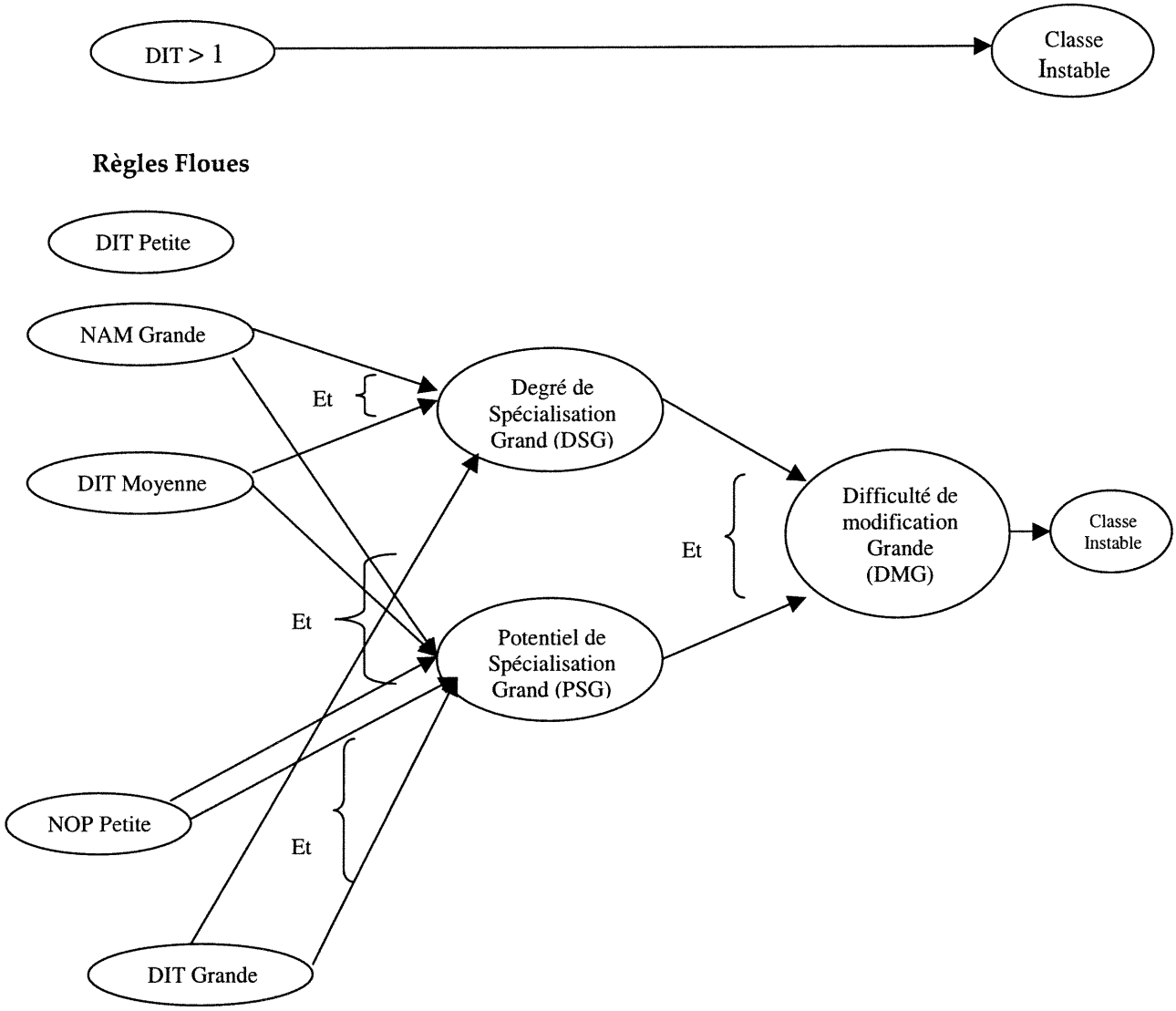


Figure 15. Dérivation des règles floues à partir d'une règle naïve et des connaissances du domaine: métrique DIT

Les notations utilisées dans les schémas de dérivation des règles présentées plus hautes décrivent un ensemble de liens de causalité. On distingue une liaison convergente lorsque, au moins, deux causes conduisent vers la même conclusion (ex : NAM grande et DIT moyenne alors Degré de spécialisation grand), une liaison divergente lorsqu'une condition participe avec d'autres pour aboutir à des conclusions différentes. Une liaison est linéaire lorsqu'une condition participe seule dans la conclusion. La signification des opérateurs logiques (ET et OU) dans le schéma se traduit par :

ET : une combinaison de deux ou plusieurs causes pour aboutir à une même conclusion.

OU : une séparation en deux ou plusieurs règles distinctes dont les conditions sont différentes et la conclusion est la même.

5.2.3 Exploitation des modèles obtenus pour la prise de décision

Nous allons appliquer le modèle de la figure 15 sur un exemple d'une classe du système Beanb.

Non de la classe	NOP	DIT	NAM	Classification
ServletDirectory	**0**	**5**	**160**	**1**

Règle naïve

(DIT=5) > 1 Alors class instable

Pour la classe **ServletDirectory** la valeur de DIT est égale à 5 donc selon la règle cette classe est instable car elle vérifie la condition.

La question qui se pose comment ont doit agir sur la conception de la classe pour la rendre stable ?

Pour le cas de la règle naïve on as pas suffisamment d'informations pour changer la conception de la classe. Pour le modèle causale les relations triviales offrent plus de détails pour agir sur la conception de la classe.

Dans le cas de la règle étendue nous ajoutons des coefficients de vérités pour instanciée la règle.

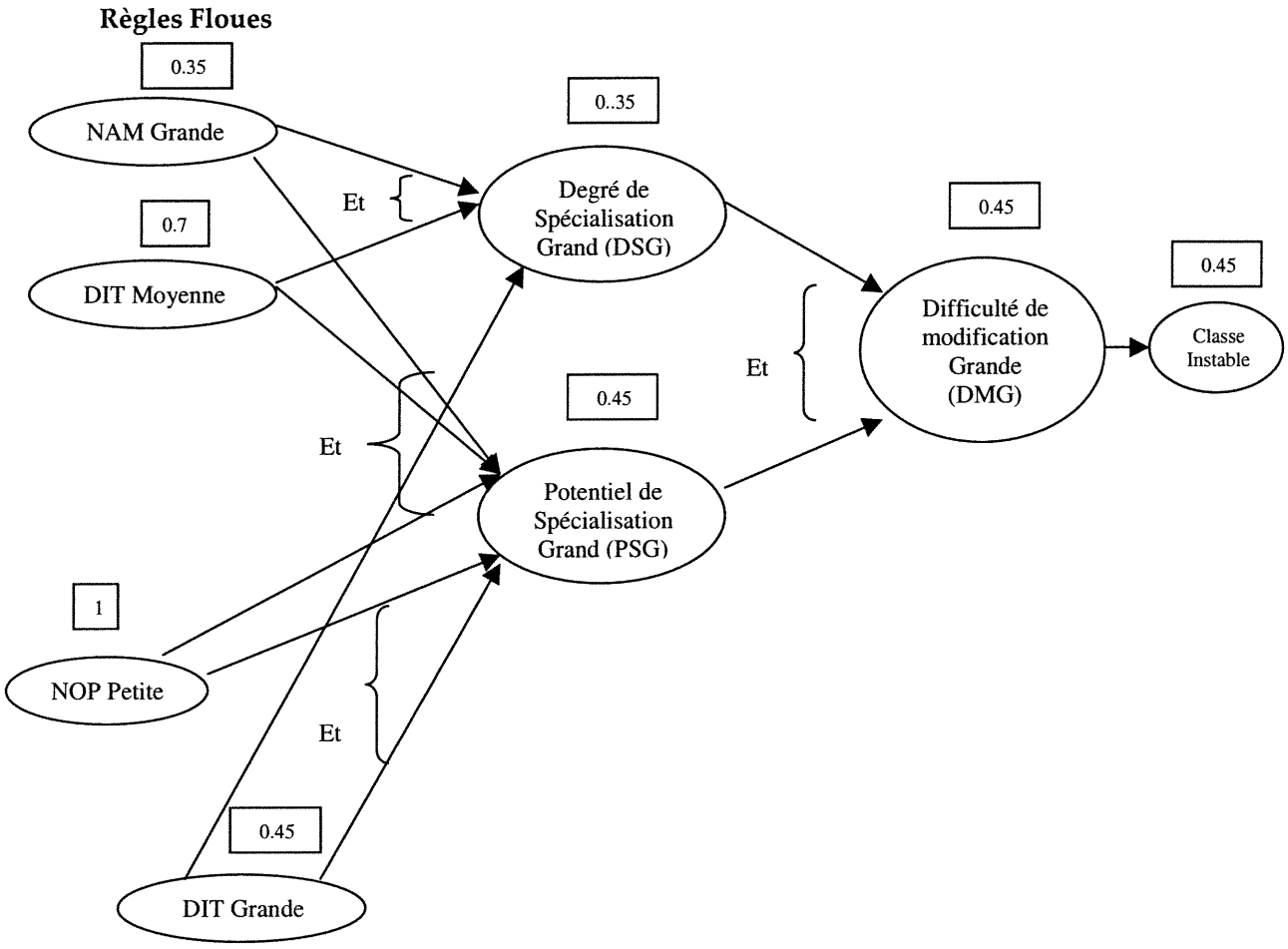


Figure 16. Affectation des valeurs de vérités pour chaque nœud du graphe présenté dans la figure précédente.

À partir du schéma étendu et en se basant sur les valeurs de vérité nous pouvons agir facilement sur les entrées. Nous suivons un raisonnement en arrière pour retracer les causes menant à la conclusion finale: classe instable.

La classe est instable cela est déduit directement de la difficulté de modification de celle ci qui est grande avec un degré de vérité égale à 0.4. La DM est causée plus par le potentiel de spécialisation que par le degré de spécialisation de la classe. Le PSG est lié aux des valeurs des deux métrique DIT grande et NOP petite.
Pour rendre la classe instable il faut diminuer la difficulté de modification de la classe en diminuant le potentiel de spécialisation. Ce dernier est lié à la diminution d'une des valeurs de métriques DIT ou NOP. La diminution de la métrique NOP signifie la création de nouvelle classes abstraite par une factorisation des classes existantes.

La prise de décision est très bien justifiée par l'existence des relations de causalité facilement explicable(triviale) et en considérant, aussi, les coefficients de vérité au niveau de chaque cause.

5.2.3 Algorithme de dérivation

Le processus de dérivation de règles floues se base sur l'éclatement des règles du modèle naïf tel que représenté dans les Figures 13,14 et15, en incluant de nouvelles causes liées par des relations facilement vérifiables par des experts. La dérivation des règles floues repose sur l'exploitation d'un ensemble de règles naïves et un ensemble de contraintes supplémentaires basées sur les heuristiques. Cette étape peut être récapitulée dans l'algorithme suivant.

Soient :

Ri : Règle
NR : Ensemble des règles naïves
CR : Ensemble des règles causales
ICR : Ensemble des règles initiales
FCR : Ensemble des règles finales

POUR *chacune de règle* ***Ri*** *dans* ***NR FAIRE***
Dériver l'ensemble des règles ***CRi*** *en se basant sur les heuristiques du domaine càd :*

a. *il existe un sous-ensemble de règles* ***ICRi*** *dont leurs conditions sont des conditions floues de type « M1 labelj »*
b. *il existe un sous ensemble de règles* ***FCRi*** *dont les conclusions sont l'estimation de la caractéristique de qualité.*
c. *les conditions de chaque règle dans* ***CRi-ICRi*** *sont des conclusions des règles dans CRi*
d. *chaque règle dans CRi représente une relation de causalité vérifiable.*

FIN

L'algorithme précédent décompose l'ensemble de règles en un ensemble de règles naïves, un ensemble de règles causales, un ensemble de règles initiales et en un ensemble de règles finales. Le principe de base est d'exécuter une succession d'itérations pour chaque règle du modèle naïf afin de les transformer en un ensemble de règles causales de sorties qui sont facilement vérifiables.

Si nous utilisons l'algorithme pour dériver la règle de la figure 14 nous aurons à suivre les étapes **a, b, c, d** suivantes :

Règle naïve :

Si OCAEC > 0 **Alors** Classe Instable.

a - Il existe un sous ensemble de règles ICRi de type *Mi Labelj*
Ex : Si DAM Moyenne **Alors** Couplage Potentiel Grand (CPG)

b - Il existe un sous-ensemble de règles FCRi dont les conclusions sont l'estimation de la caractéristique de qualité.
Ex : Si le risque de l'effet de changement secondaire grand (RECG) **Alors** Classe instable

c - La condition des règles de CRi-ICRi sont des conclusions des règles dans CRi
Ex : La condition : le risque de l'effet de changement secondaire grand (RECG) de la règle précédente représente une conclusion dans une autre règle (appartenant à CRi-ICRi); les règles sont liées entre elles.

c - Toutes les règles de CRi représentent des relations de causalité vérifiables par un expert du domaine en génie logiciel.

5.2.4 Application de l'algorithme pour la dérivation des règles causales

Nous appliquons l'algorithme décrit plus haut pour dériver manuellement les règles causales. Nous obtiendrons la base de règles qui apparaît plus bas et qui est la traduction des Figures 13,14 et 15. Plusieurs langages existant permettant de définir des règles selon une grammaire bien déterminée. Un exemple est la grammaire BNF qui définit une syntaxe de représentation des règles. Dans notre cas, nous nous sommes inspirés de la syntaxe des règles générées par l'outil d'apprentissage C4.5 (voir l'exemple de règle classique au Chapitre 4, Section 5) pour définir la syntaxe des règles floues. Nous avons conçu notre base de règles de façon à pouvoir vérifier les deux critères suivants: la cohérence et la complétude. La cohérence d'une base de règle est définie comme l'élimination de l'ensemble des règles redondantes, dupliquées, subsumées, inutiles, inatteignables, circulaires et des règles en conflit. La vérification de la complétude consiste à détecter dans une base de règle la présence de littéraux inutiles et de valeurs manquantes.

Base de règles floues :

Règles de la Figure 12 :

Rule1 :
DAM Moyenne
➔ Couplage Potentiel Grand

Rule2 :
DAM Petite
➔ Couplage Potentiel Grand

Rule3 :
Couplage Potentiel Grand
➔ Effet négatif de changement Grand

Rule4 :

Effet négatif de changement Grand

➔ Classe Instable

Règles de la Figure13 :

Rule1 :
OCAEC Grande
➔ Impact de Couplage Grand

Rule2 :

OCAEC Moyenne
NAA Petite
➔ Effet de modification d'un attribut Grand

Rule3 :
Effet de modification d'un attribut Grand
➔ Effet négatif de changement Grand

Rule4 :

Effet négatif de changement Grand

➔ Classe Instable

Règles de la Figure14 **:**

Rule1:
NAM Grande
DIT Moyenne
➔ Degré de Spécialisation Grand
Rule2 :
DIT Grande
➔ Degré de Spécialisation Grand
Rule3 :
NAM Grande
DIT Moyenne
NOP Petite
➔ Potentiel de Spécialisation Petit
Rule4 :
DIT Grande
NOP Petite
➔ Potentiel de Spécialisation Petit
Rule5 :
Potentiel de Spécialisation Petit
Degré de Spécialisation Grand
➔ Difficulté de Modification Grande
Rule6 :
Difficulté de Modification Grande
➔ Classe Instable

Les règles obtenues par le mécanisme d'éclatement représentent, chacune, des connaissances exprimées sous la forme: (**si** condition(s) **alors** conclusion(s)). Ces règles seront utilisées par le moteur d'inférence (MI) pour déduire les conclusions selon un mode de raisonnement à chaînage avant, arrière, ou mixte. Les règles dérivées ont une syntaxe restreinte; ce sont des règles enchaînées de telle sorte que la condition d'une règle représente la conclusion d'une autre règle. Le déclenchement de ces règles se fait par l'examen et la comparaison faite par le MI entre les prémisses des unes et les conclusions des autres. La conception d'une base de règles par blocs regroupant des règles liées entre elles donne une bonne structure et une meilleure lisibilité à la base de règles. Cette stratégie décrit une organisation logique de la base de règle, facilitant ainsi aux experts le développement d'une vue partielle par module de celle-ci.

Les règles causales obtenues en se basant sur l'algorithme défini plus haut sont caractérisées par des conclusions finales (classification) ou intermédiaires, par des

conclusions intermédiaires qui pourrons être des entrées d'une ou de plusieurs règles, par des prémisses en commun ou elles concluent sur une même conclusion.

Le mécanisme de dérivation de règles causales est difficile à automatiser. Une solution peut être la création d'un dépôt contenant l'ensemble de relations causales entre les différents attributs de logiciels, qui remplaçant l'intervention des experts dans la construction du modèle basé sur des heuristiques du domaine. Une recherche automatisée des relations appropriées entre les attributs de logiciel dans la base peut servir alors pour dériver les règles causales du modèle.

5.3 INFÉRENCE DES RÈGLES FLOUES

Nous présentons dans cette section la troisième étape de notre processus de modélisation flou qui est l'étape d'inférence. Pour ce faire, nous avons développé un outil qui permet entre autre, de faire l'inférence des modèles de qualité. Nous expliquons par la suite la présence de l'étape de fuzzification du processus d'inférence, cette dernière faisant la différence entre les MI flou et classique.

5.3.1 Moteur d'inférence flou

La nécessité d'utiliser un MI flou permettant d'évaluer les règles floues du modèle proposé (base de règles et leurs fonctions d'appartenance associées). Nous à guider vers le développement d'un outil « MI flou ».

Le MI est un programme qui met en œuvre un mécanisme d'interprétation des règles. La stratégie d'un MI comporte trois phases successives: la détection des règles candidates, la sélection d'une règle à appliquer et le déclenchement des règles retenues.

Le MI prend comme entrée les données fuzzifiées et les règles floues du système. Le MI le plus couramment utilisé est celui dit de **Mamdani** [33]. Il présente une simplification du mécanisme général basé sur ''**l'implication floue**'' et le ''**Modus Ponens Généralisé**'' (pas de clause OU dans les conclusions).

Une base de règles floues de Mamdani comprend donc des règles linguistiques faisant appel à des fonctions d'appartenance pour décrire les concepts utilisés. Dans notre cas, elles s'appliquent à l'ensemble des métriques liées à la stabilité. Inférer une règle c'est la choisir pour évaluation une fois que ses prémisses ont été évaluées. On peut alors déterminer la vérité de ces conclusions.

L'outil (Moteur d'Inférence) développé dans le cadre du projet OO1 effectue les opérations suivantes:

✓ Analyser les règles d'inférences.
✓ Utiliser une stratégie de chaînage mixte pour inférer les règles.
✓ Inclure l'étape de fuzzification dans le processus d'inférence (détermination des fonctions d'appartenance).

Le mécanisme d'inférence comprend les étapes suivantes:

- La fuzzification qui consiste à évaluer les fonctions d'appartenance utilisées dans les prédicats des règles.

- L'activation: le degré d'activation d'une règle est l'évaluation du prédicat de chaque règle par combinaison logique des propositions du prédicat. Dans ce cas, le ET est réalisé en effectuant le minimum entre les degrés de vérité des propositions.

- L'implication est le degré d'activation de la règle qui permet de déterminer la conclusion de celle ci. L'ensemble flou des conclusions est construit en réalisant le minimum entre le degré d'activation et la fonction d'appartenance

- L'agrégation: l'ensemble flou global de sortie est construit par agrégation des ensembles flous obtenus par chacune des règles concernant cette sortie.

Le mécanisme de déclenchement des règles dans un MI flou utilise trois stratégies de chaînage pour inférer les règles, à savoir chaînage avant, le chaînage arrière et le chaînage mixte. Le principe du chaînage est expliqué par une chaîne d'inférence ou chemin de Raisonnement.

Le chaînage avant: le raisonnement est guidé par les données; si la liste des prémisses d'une règle est vérifiée, la règle est déclenchée et ses conditions sont ajoutées à la base de faits.

Le chaînage arrière: on part d'une conclusion que l'on désire prouver et l'on ne déclenche que les règles capables d'établir cette conclusion. Le type de raisonnement en chaînage arrière est guidé par les buts hypothétiques à atteindre; le système alors recherche une façon pour prouver ces buts.

Le chaînage mixte est une succession de chaînages avant et arrière. Le chaînage avant appel le chaînage arrière lorsqu'il trouve une conclusion intermédiaire. Le chaînage arrière appel le chaînage avant lorsqu'il cherche les résultats dans les règles (la valeur de vérité).

Le MI en chaînage avant opère par cycles. À chaque cycle, il examine l'état actuel de sa base de faits et déclenche les règles dont les prémisses sont satisfaites, ajoutant les nouvelles conclusions qu'il en tire à sa base de données. Le processus s'arrête lorsqu'on ne peut plus déclencher de règles.

Le MI flou applique les règles floues, et fournit une ou plusieurs sorties floues qui peuvent être réinjectées dans le processus d'inférence. Notre MI garde les traces d'exécutions du système pour une éventuelle utilisation dans La restructuration. Cependant l'inférence des règles se fait pour chaque classe de l'application à part.

5.3.2 Etape de fuzzification dans le MI

À la différence des moteurs d'inférence classiques, les moteurs d'inférence flous ajoutent les étapes de fuzzification et de défuzzification dans le processus d'inférence. Ces dernières représentent l'interface entre le monde extérieur et le système flou. L'étape de fuzzification prend comme entrée l'ensemble de données mesurées (métriques de stabilité et leurs fonctions d'appartenances). La défuzzification représente une sortie de notre système permettant de passer des valeurs floues aux valeurs réelles.

5.4 DÉFUZZIFICATION

L'étape de défuzzification représente la dernière phase de notre processus flou. Cette phase est expliquée par la transformation des valeurs floues de sorties en valeurs concrètes. La caractéristique prédite est la stabilité qui prend les valeurs Stable ou Instable.

Si nous utilisons une métrique continue de la stabilité (à définir), nous établirons sa fonction d'appartenance qui sera utilisée pour faire la défuzzification à l'aide d'une des techniques déjà citées dans le chapitre 3 (section 2). La technique de défuzzification la plus couramment utilisée est celle du centre de gravité mais cette dernière n'est pas applicable aux variables linguistiques discrètes.

Nous détaillerons dans le chapitre qui suit la partie implémentation et nous traduisons l'application des étapes de notre approche de construction, exploitation et validation de modèle prédictif proposé par une utilisation de l'outil développer à ces fins.

Implémentation

Ce chapitre décrit la partie réalisation de notre projet et les difficultés rencontrées au niveau de l'implémentation. Dans la première section, nous décrivons les techniques et les outils choisis afin de concevoir l'architecture proposée et les motivations de ces choix. Dans la deuxième section, nous décrivons l'architecture du système en termes de fonctionnalités et de modules. Tout au long de la troisième section, nous verrons pour chaque module les principales classes d'objets développées ainsi que les concepts utilisés tels que le chargement dynamique des objets à partir d'une base de donnée relationnelle ou sont stockées les données de notre système. Nous abordons aussi les détails d'implémentation de notre système et nous décrivons chacun des modules constituant l'outil: parseur (analyseur) de la base de règles, inférence de règles, fuzzification, restructuration, stockage de données et génération des rapports. Dans la quatrième, section nous décrivons les fonctionnalités de l'outil OO1 et nous présentons des exemples d'exécution avec des écrans de saisies manipulant les fonctionnalités du système. Enfin nous proposons dans la conclusion les points forts de l'outil et les améliorations futures.

6.1 ENVIRONNEMENT DE TRAVAIL

6.1.1 Motivations

L'utilisation d'une conception objet exprimée sous forme de diagrammes UML pour décrire la partie statique de notre système et l'utilisation d'un langage java sont les points forts de notre architecture. Ils offrent des possibilités de réutilisation, la souplesse de manipulation de données, une bonne manipulation des interfaces et des possibilités de stockage de données du système grâce au JDBC.

Nous avons opté pour développer notre système d'utiliser le langage java. Ce langage présente les avantages d'être portable, orienté objet, et multi-threads. Il offre des

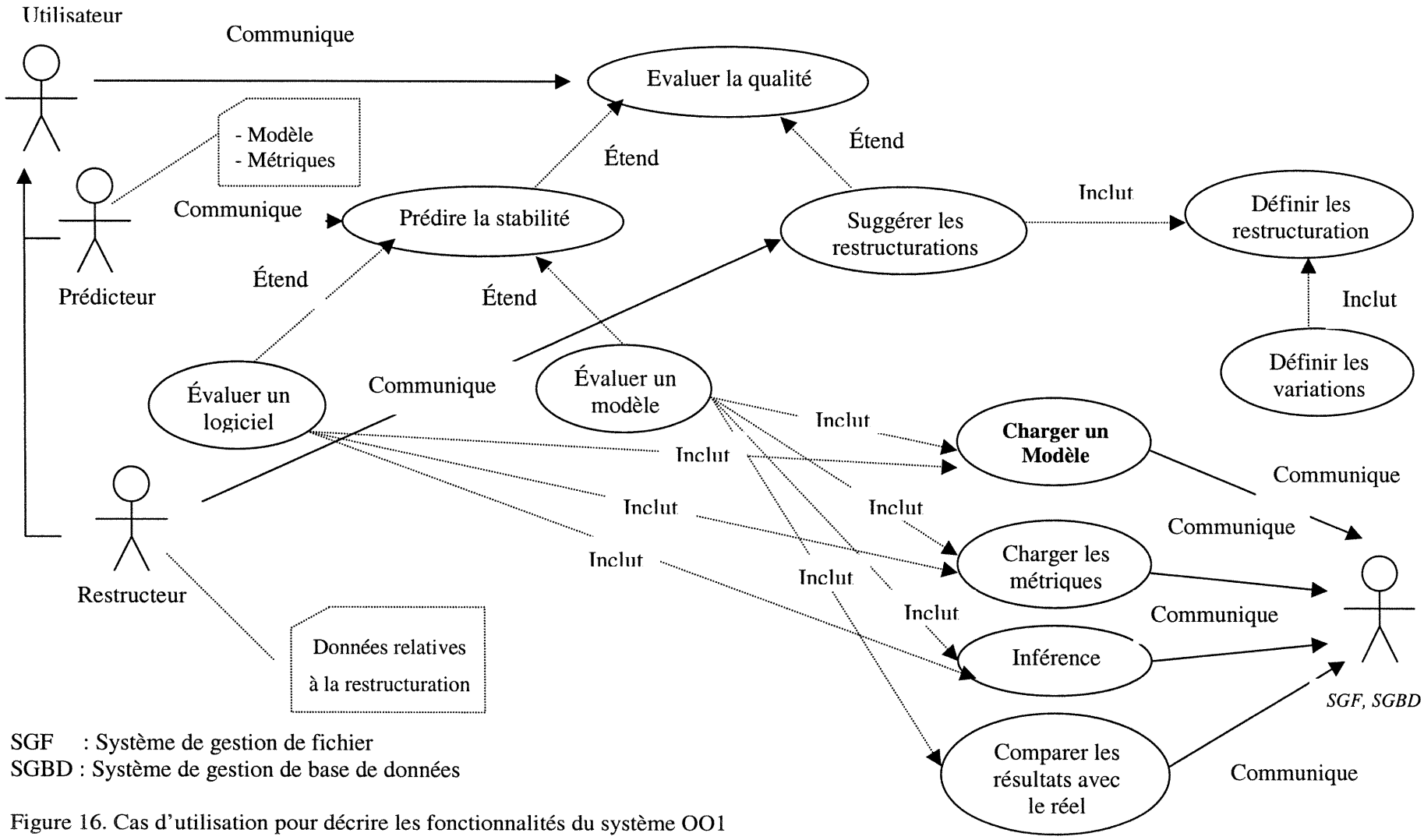
fonctionnalités intéressantes pour faciliter la distribution dynamique du code et le transfert des objets. Ce dernier est indépendant de la plate-forme et respecte les concepts de la programmation orientée objet. Il offre un ensemble d'API adaptés à différents besoins (bases de données, sécurité, applications distribuées, réseaux, interfaces, etc.).

La conception de notre système est réalisée à l'aide du langage de modélisation UML. Ce dernier est un langage puissant dont la notation graphique permet d'exprimer visuellement une solution objet. L'aspect formel de sa notation limite les ambiguïtés et les incompréhensions. Son aspect visuel facilite la comparaison et l'évaluation de solutions. Son indépendance par rapport aux langages d'implémentation, au domaine d'application et au processus en fait un langage universel. Les diagrammes de classes conçus par UML seront traduits sous forme de classes, écrites en java, qui décrivent le contenu du système en terme de données et de comportement.

6.2 ARCHITECTURE GÉNÉRALE DU SYSTÈME

6.2.1 Description de l'architecture en fonctionnalités :

Le travail que nous avons élaboré s'inscrit dans le cadre d'un projet global comprenant deux missions principales: l'évaluation de la qualité d'un logiciel et la proposition des alternatives de restructuration selon deux approches, classique et floue. Nous avons proposé une architecture générale décrivant les principales fonctionnalités du système et les relations entre elles. Cette architecture est décrite par le diagramme de **cas d'utilisation** UML suivant :



SGF : Système de gestion de fichier
SGBD : Système de gestion de base de données

Figure 16. Cas d'utilisation pour décrire les fonctionnalités du système OO1

6.2.2 Description de l'architecture en termes de modules :

Notre système est composé d'un ensemble de modules qui répondent à un ensemble de fonctionnalités décrites dans la Figure 16. Ces modules sont représentés comme suit :

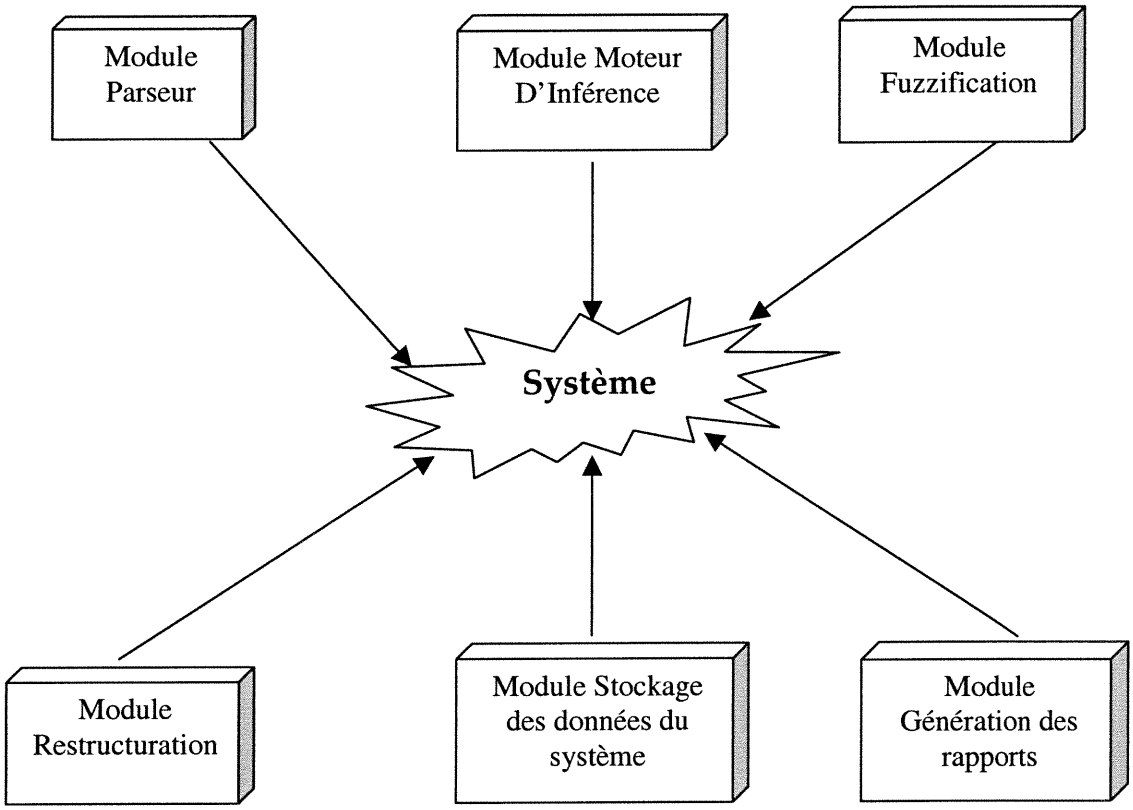


Figure 17 : Architecture générale du système en modules

- Module parseur: englobe toutes les classes et interfaces relatives à l'analyse d'une base de règles floues ou classiques.

- Module moteur d'inférence: contient les classes et interfaces permettant d'évaluer un modèle (utiliser le MI pour inférer les règles sur des jeux de données) selon de mode classique ou floue.

- Module fuzzification: englobe les classes et interfaces décrivant le processus de fuzzification (représentation des fonctions d'appartenance, des valeurs de métriques et calcul des valeurs de vérités).

- Module restructuration: décrit les classes et les interfaces relatives au processus de restructuration pour un modèle classique ou flou.

- Module stockage: regroupe les classes et interfaces permettant de stocker l'ensemble des données du système (fonction d'appartenance, données de métriques et bases de règles) dans une base de donnée.

- Module génération des rapports: présente les classes et interfaces permettant la génération des rapports écrits dans des pages XML. Ces derniers contiennent des résultats de prédiction, d'évaluation et des suggestions de restructurations.

6.3 DESCRIPTION DE L'IMPLÉMENTATION DES MODULES DU SYSTÈME.

L'implémentation des modules représentant notre système englobe:

- L'implémentation des fonctionnalités majeures du système sous forme de modules: parseur, inférence, fuzzification, évaluation et restructuration,
- Le stockage des données utilisées pour exécuter les fonctionnalités de l'outil,
- La conception et le développement des interfaces graphiques de l'outil.

Les modules de l'outil OO1 sont implémentés entièrement en java pour les raisons citées au début de ce chapitre. La base de donnée utilisée est une base de données Access sous Windows.

Dans ce qui suit, nous allons décrire en détail les modules qui se rattachent à ce travail et qui sont les modules parseur, prédiction, moteur d'inférence, fuzzification, Stockage de donnée et génération des rapports.

6.3.1 Module parseur

Ce module décrit l'étape d'analyse d'un modèle de qualité. Il présente l'ensemble de classes qui analyse une base de règle classique ou floue. Ce module décompose une règle classique ou floue en deux parties droite et gauche. La partie droite contient les conditions de la règle et la partie gauche contient la conclusion. Une condition peut être décomposée en termes linguistiques et en une variable linguistique.

Le module parseur contient une classe principale, parseur, qui constitue l'exécutif de l'analyseur; elle contient la méthode **parserRule()** responsable de déclenchement du processus. L'interface **model** sépare le parsing d'un modèle classique du parsing d'un modèle flou.

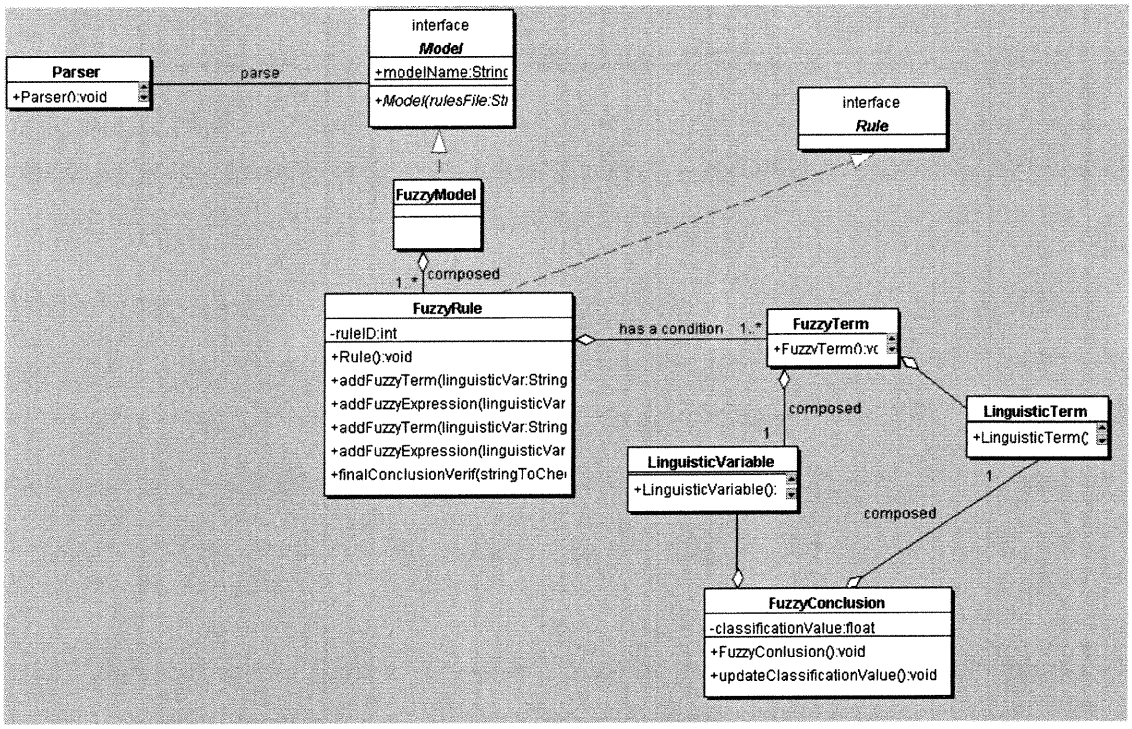


Figure 18 : Diagramme de classe UML du module parseur flou

6.3.2 Module moteur d'inférence

Le module moteur d'inférence décrit l'étape de déclenchement des règles. Ce module est composé de la classe mère **InférenceEngine** qui est responsable de déclenchement du processus de l'inférence. Cette classe contient un ensemble de méthodes pour l'inférence des règles et la sauvegarde des traces d'exécution du système. La classe **FactBase** implémente l'ensemble des méthodes qui manipulent une base de faits. Deux classes distinctes héritent de cette classe : **ClassicFactBase** et **FuzzyFactBase**. Ces deux classes disposent des outils nécessaires pour différencier l'inférence d'un modèle flou de l'inférence d'un modèle classique. Le diagramme de classe du module Moteur d'Inférence décrit l'ensemble de classes impliquées dans ce processus d'inférence est se présente comme suit :

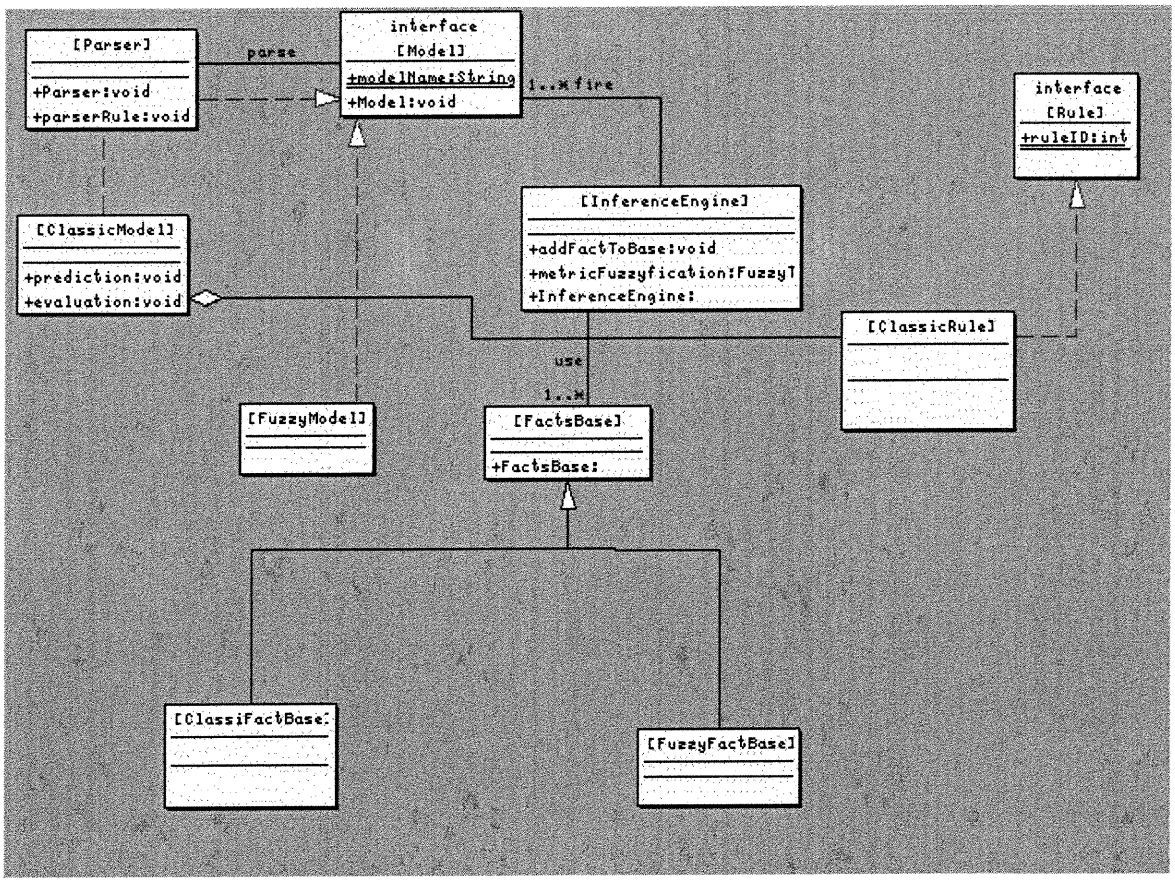


Figure 19 : Diagramme de classe UML du module moteur d'inférence

6.3.3 Module fuzzification

L'étape de fuzzification et une étape délicate dans le processus de modélisation floue. L'implémentation de cette étape et son intégration dans le système global sont expliquées dans cette partie. Le diagramme de classe de la partie fuzzification présente l'ensemble de classes qui décrivent l'aspect statique de la fuzzification (Voir la Figure 20). On crée dynamiquement les instances de la classe **InputMetric** lorsqu'on récupère les données (valeurs de la métrique) de la base de données. La classe **MemberShipfunction** est composée de 5 classes qui représentent les formes de fonctions d'appartenances existantes (**Trapezoidal**, **Triangular**, **LeftTrapezoidal**, **RigthTrapezoidal** et **Singleton**). Les instances de ces dernières sont aussi créées dynamiquement une fois avoir récupéré les données sur la fonction d'appartenance stockées dans la base de donnée. Cette technique est dite la réflexion en java.

Chaque forme de fonction d'appartenance est représentée par un ensemble de points qui définissent les limites de la courbe de la fonction d'appartenance. Par exemple la forme triangulaire est définie par 3 points, et la forme trapézoïdale par quatre points.

Les données sur les fonctions d'appartenance sont stockées dans une base de données Access. Ces données représentent la forme de la FA et les points qui définissent ses bornes.

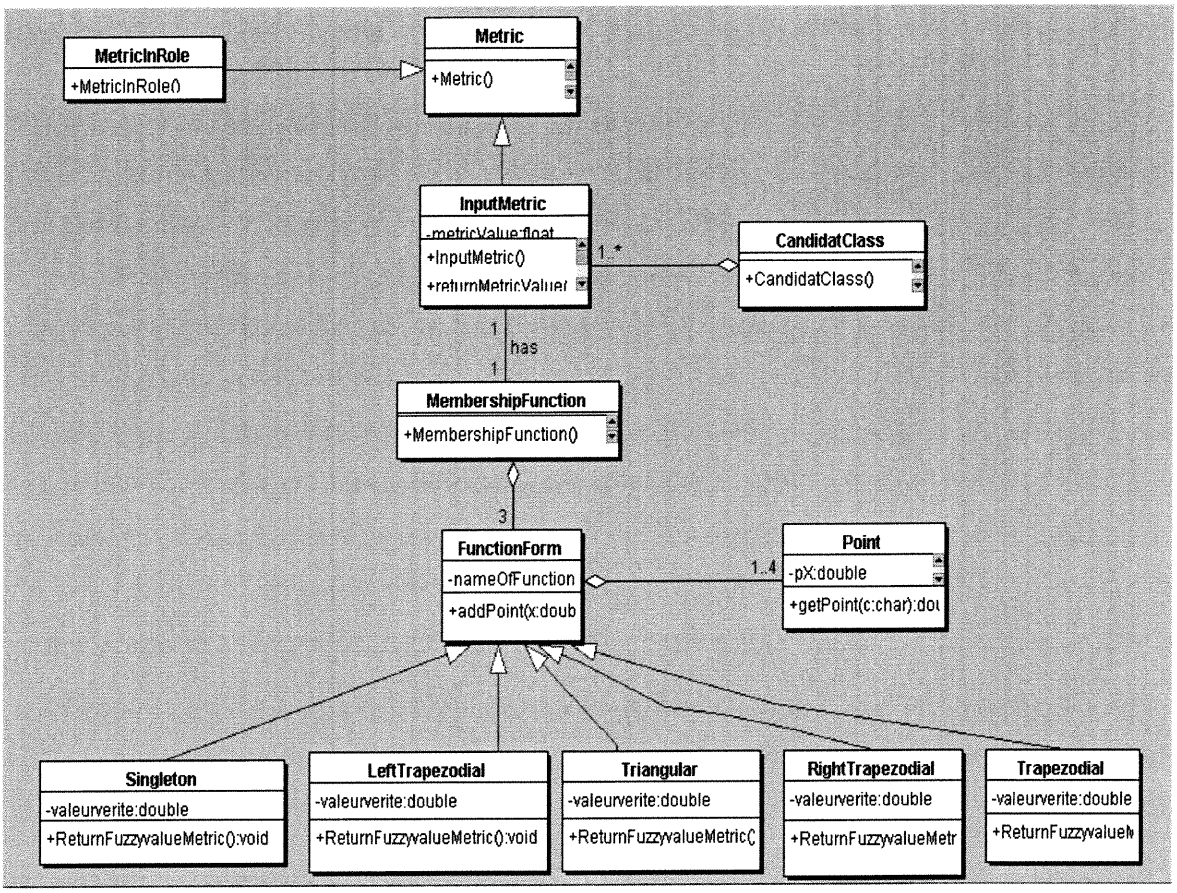


Figure 20 : Diagramme de classe UML du module fuzzification

6.3.4 Module stockage de données

Ce module présente les techniques du stockage utilisées pour rendre persistantes les entrées de notre système. Les entrées du système sont stockées dans une base de donnée. Ces dernières représentent:

- Les données des métriques,
- Les fonctions d'appartenance,
- Les relations métriques – restructuration,
- La base de règles.

Le diagramme entité-association représentant les différentes entités de stockage est comme suit :

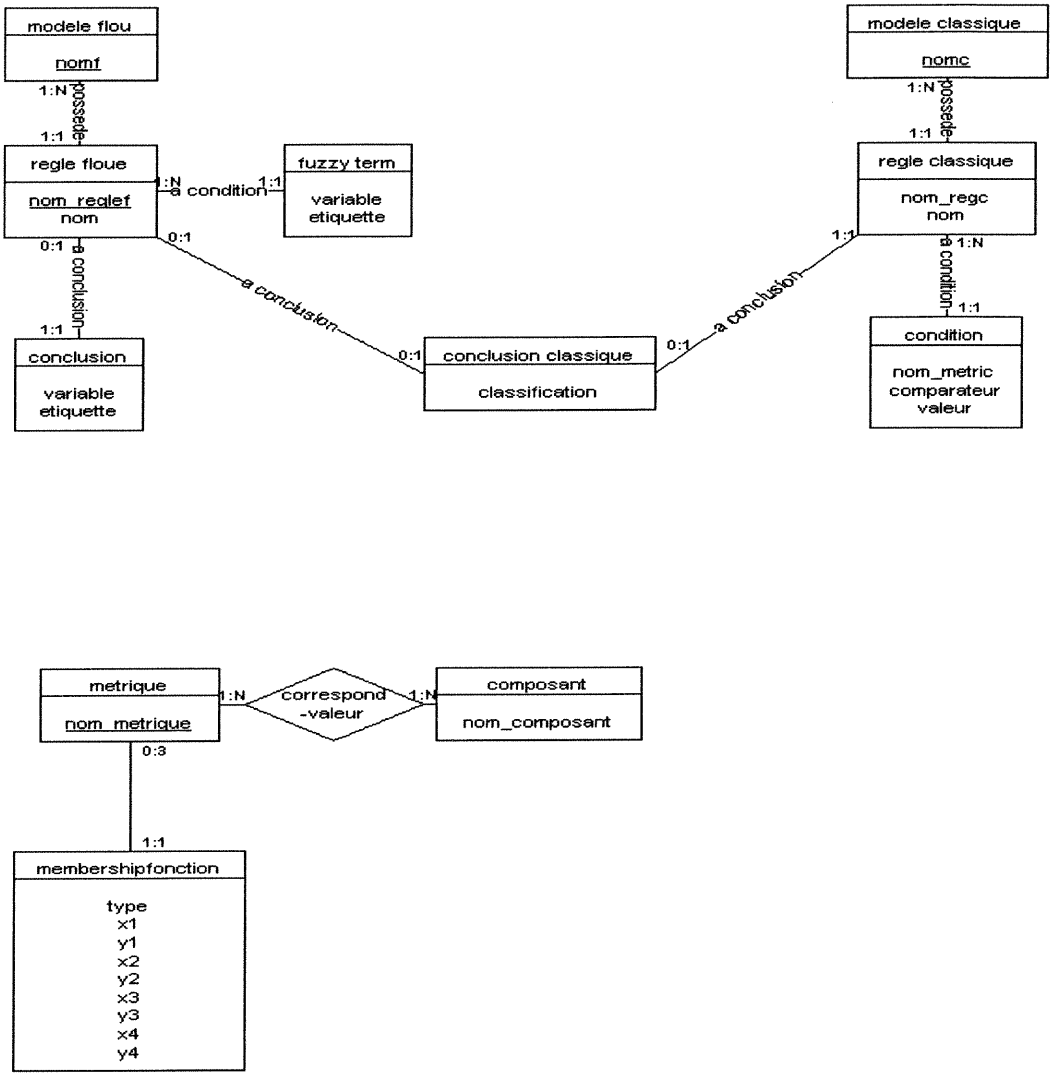


Figure 21 : Diagramme entité association du module stockage

Le diagramme entité-association décrit les principales entités utilisées dans la base de données et leurs relations. Ce diagramme est découpé en deux parties : une partie qui décrit la base de règles et une partie qui décrit les données des métriques et les données des fonctions d'appartenance.

La partie base de règles: un modèle flou est représenté par l'entité m**odèle flou,** ce dernier est composé d'un ensemble de règles floues décrites dans l'Entité **règle floue.** Cette entité est composée des termes linguistiques (entité **fuzzy term**) et des conclusion (entité **conclusion** ou entité **conclusion classique**). Un modèle classique est représenté par l'entité m**odèle classique;** ce dernier est composé d'un ensemble de

règles classiques décrites dans l'entité **règle classique.** Cette entité est composée des conditions (entité **condition**) et des conclusions (entité **conclusion classique**).

La partie données des métriques et des fonctions d'appartenance: Le nom de la métrique est indiqué dans l'entité **métrique** et sa valeur correspondante figure dans l'entité **composante.** La fonction d'appartenance d'une métrique est décrite dans l'entité **MemberShipfunction** qui contient le type de la fonction (trapèze, triangle, etc.) et les points qui définissent cette fonction.

Nous présentons la description statique du module stockage de données sous forme d'un diagramme UML. Ce dernier décrit les principales classes utilisées pour stocker les données de notre système, qui sont: les données sur la base de règles classique et floue, et les données des métriques et des fonctions d'appartenance.

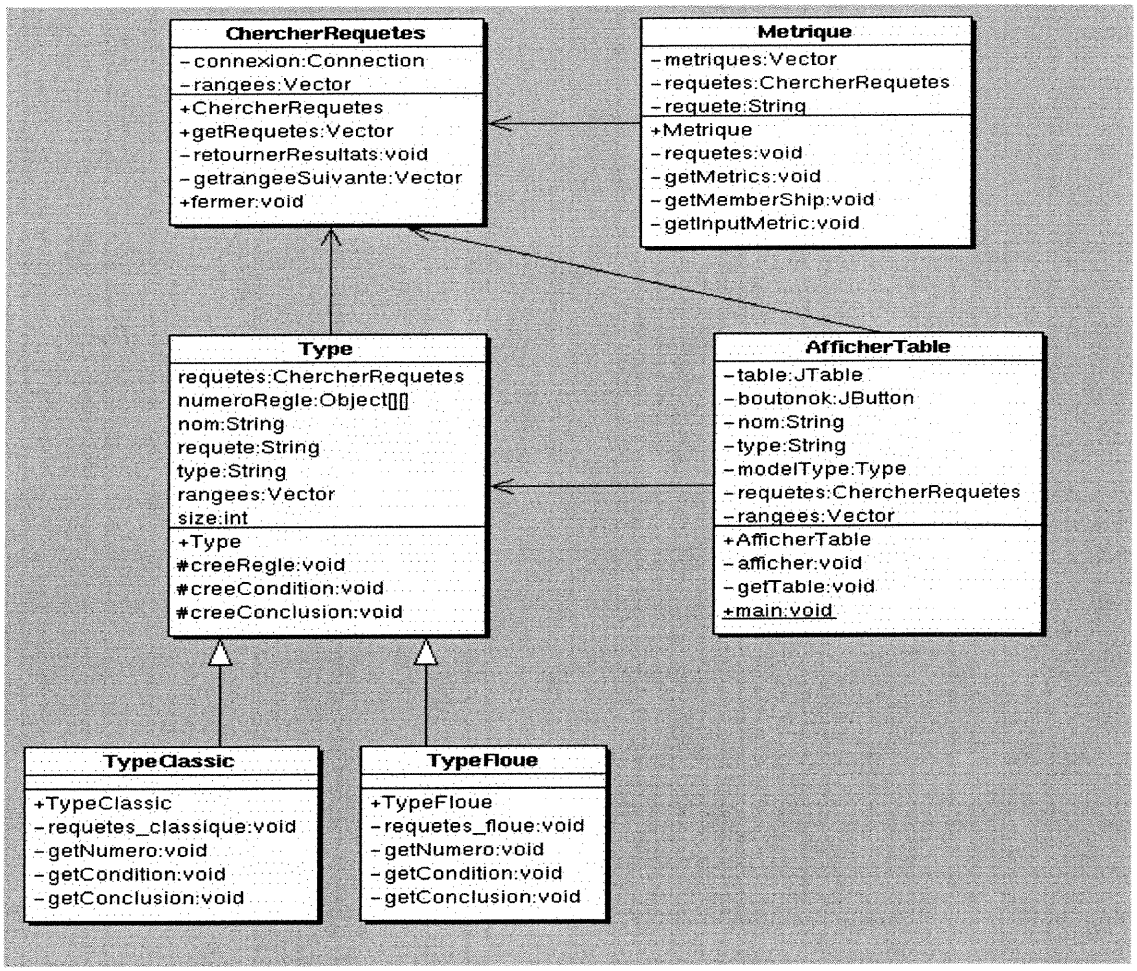


Figure 22. Diagramme de classe du module stockage

Ce diagramme décrit les principales classes du module stockage de données. Ce dernier est composé de la classe **ChercherRequetes** qui contient un ensemble d'outils pour faire la connexion avec la base de donnée, chercher les données dans la base, récupérer les résultats des requêtes et fermer la connexion avec la base de donnée. La classe **Metrique** contient les outils pour manipuler les données des métriques (Nom, valeur et fonction d'appartenance). La classe **Type** est une classe contenant des informations sur les bases de règles et manipulant les méthodes responsables de créer une règle, créer une condition et créer une conclusion. Les deux classes **TypeClassic** et **TypeFloue** héritent de la classe **Type** et contiennent les méthodes nécessaires pour stocker les données d'une base de règles floues et celles d'une base de règles classiques. La classe **AfficherTable** est la classe qui contient la méthode **main().**

6.3.5 Module génération de rapport

Le module génération de rapport décrit la technique utilisée pour générer des rapports. Les rapports générés contiennent des informations concernant suivantes:

- Description du système à étudier (nombre de classes, nom des métriques impliquées) et du modèle à évaluer.
- Résultats de prédiction.
- Résultats d'évaluation.
- Résultats de restructuration.
- Des indicateurs statistiques (moyenne, médiane, variance, etc.) de qualité extraits des la bibliothèque de classes.
- Des représentations sous forme d'histogrammes pour chaque métrique.
- Calcul de deux métriques de qualité portant sur l'exactitude et la complétude.

(Voir le rapport de prédiction, d'évaluation et de restructuration joint en annexe).

6.4 DESCRIPTION DE L'OUTIL (OO1)

L'outil développé dans le cadre du projet vise l'automatisation de deux aspects de la qualité de logiciel. La première porte sur l'automatisation du processus de prédiction de la qualité de logiciel et la deuxième traite l'aspect restructuration automatique de logiciel [34].

La majorité des étapes de l'approche ont été implantée dans le cadre de deux travaux de recherche sur la qualité. Nous avons intégré les deux travaux dans un environnement homogène représentant l'outil "**OO1**".

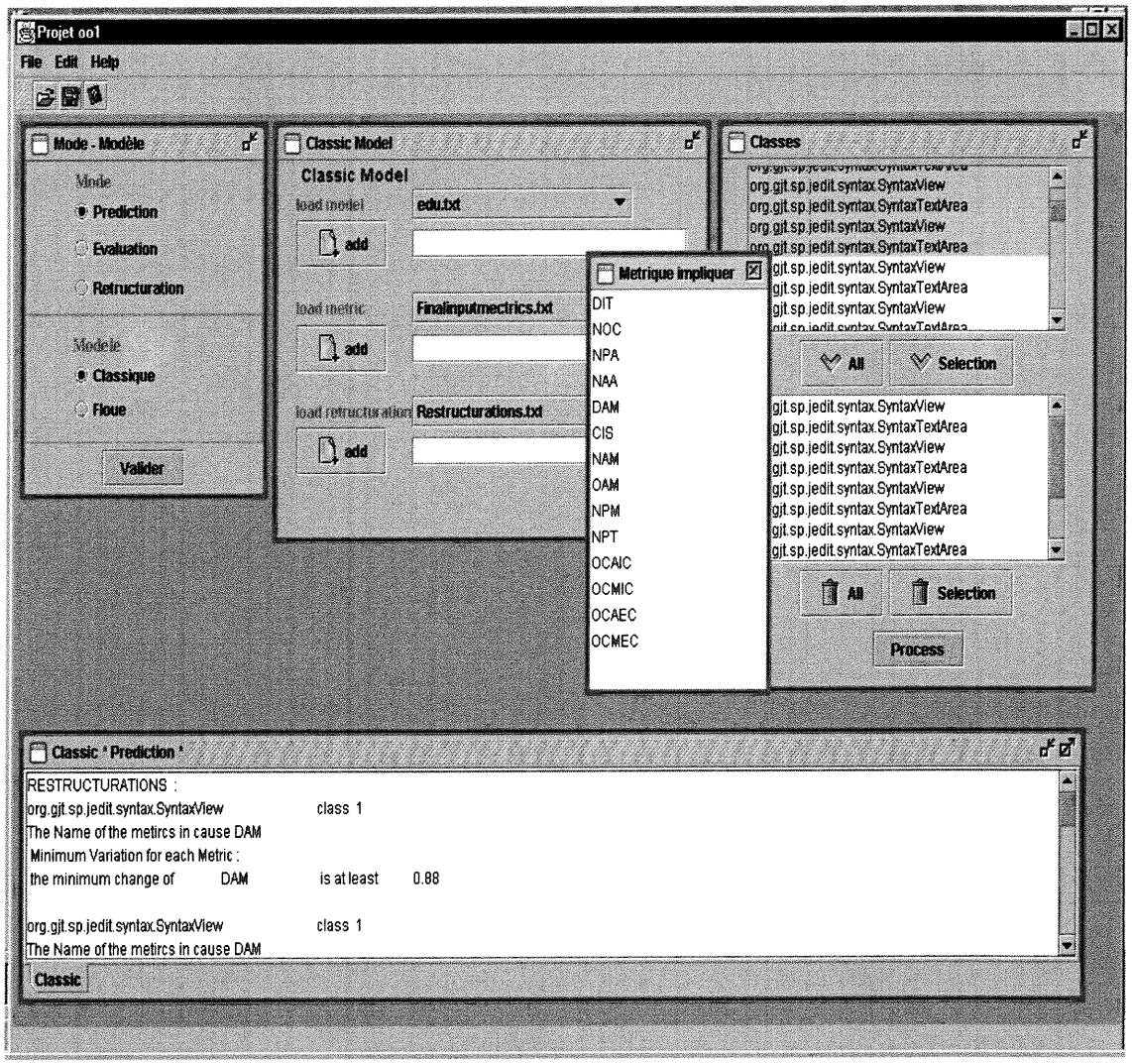


Figure 23 : Interface principale de l'outil OO1

L'interface principale est composée de trois fenêtres qui sont comme suit:

Interface mode-modèle: cette interface propose à l'utilisateur le choix du mode d'évaluation qui peut être une prédiction, une évaluation ou une restructuration. Elle propose aussi à l'usager le choix de mode de prédiction (classique ou flou). Cette opération est validée par le bouton valider.

Interface type du modèle: cette interface permet le chargement d'un modèle, le chargement des données de métriques, le chargement des données de fonctions d'appartenance et enfin le chargement des données de restructurations. Toutes ces opérations seront validées par le bouton valider.

Interface classes: présente les noms des classes choisies d'un système. Le bouton ***Process*** est responsable du déclenchement d'une prédiction, évaluation ou restructuration.

Interface message: représente une fenêtre pour l'affichage des résultats de prédiction, d'évaluation ou de restructuration.

Certaines particularités du système implanté sont justifiées par la possibilité offerte à l'usager de changer son choix à n'importe quel moment et dans n'importe quelle situation et d'exécuter les autres fonctionnalités du système tant que le bouton *Process* n'est pas enfoncé. Par exemple, il peut changer le mode ou le model en cours du traitement, ce qui rend le système plus facile à utiliser et plus souple à manipuler (l'usager ne se trouve pas bloqué devant un seul choix ou une seule proposition).

6.4.1 Fonction prédiction

La fonction prédiction permet d'évaluer la qualité d'un logiciel selon les approches classique et floue. Cette étape permet d'évaluer un modèle sur un échantillon de données constitué d'un ensemble de métriques pour chaque classe du système, cela consiste à:

- Charger un modèle (base de règles) à partir d'une base de données.
- Charger les métriques extraites sur les applications qu'on estime évaluer à partir d'une base de données : les valeurs de métriques et les fonctions d'appartenances pour l'approche floue.

- Appliquer le modèle sur les métriques.

L'étape de prédiction est déclenchée par l'utilisateur selon le choix d'un modèle (classique ou flou). L'usager charge les éléments nécessaires pour une prédiction qui sont : le modèle (base de règles), les valeurs de métriques, les données de restructurations et les données sur les fonctions d'appartenance. Une liste de classes apparaît dans la fenêtre classes offrant la possibilité de sélectionner les classes à évaluer. Le bouton *Process* permet de lancer cette opération. Ensuite, les résultats de prédiction s'affichent sur une fenêtre à part. La figure suivante présente l'enchaînement détaillé de ces étapes de prédiction.

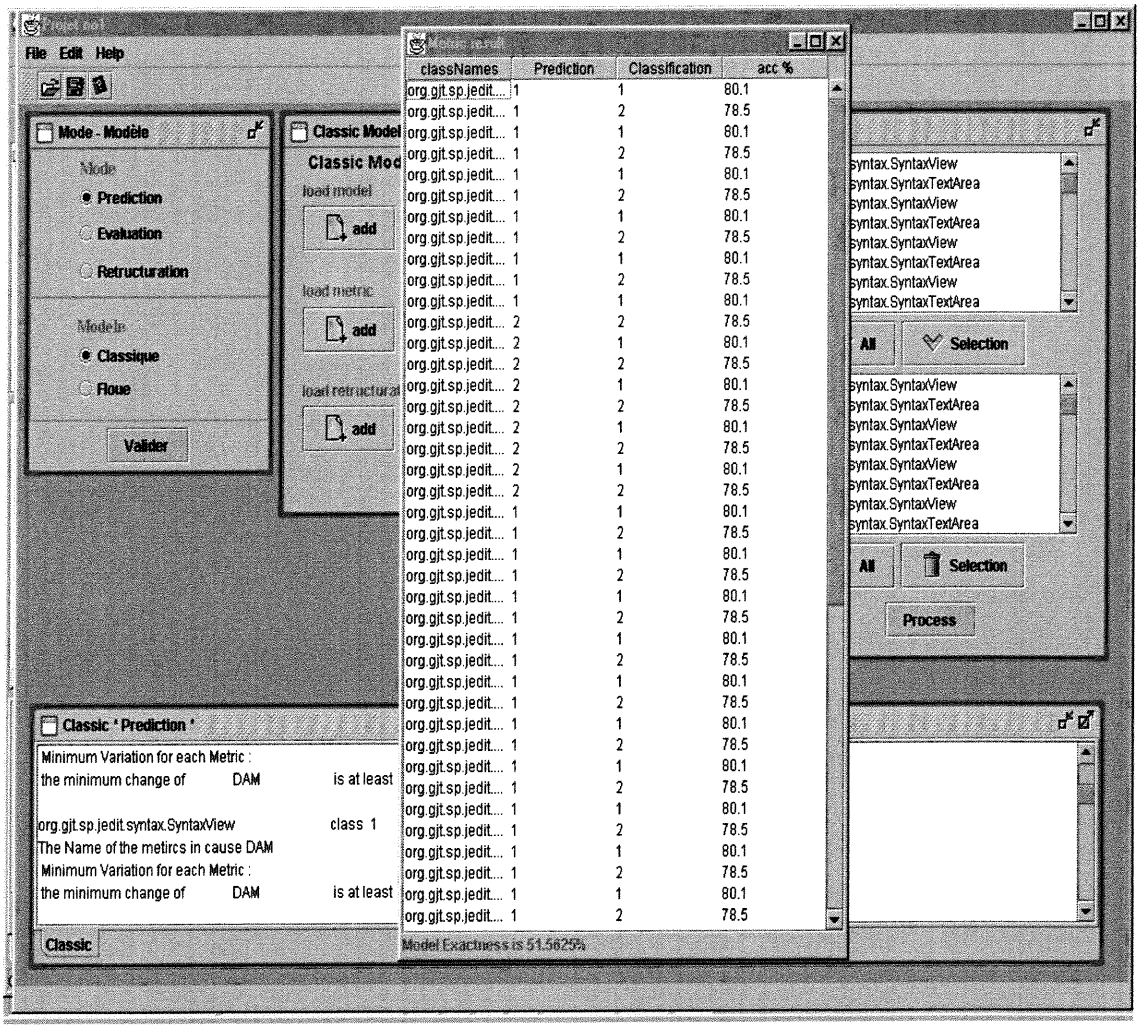


Figure 24 : Processus de prédiction d'un modèle classique

6.4.2 Fonction restructuration

Ce module propose une technique pour détecter automatiquement des situations où une restructuration particulière peut être appliquée pour améliorer la qualité d'un système. Le processus de détection est basé sur l'analyse de l'impact de plusieurs restructurations sur des métriques logicielles. La restructuration est alors pilotée par la sélection des variations dans le but d'éviter des situations d'anomalies [34].

Le déclenchement d'un processus de restructuration suit les mêmes étapes que celles du processus de prédiction. Le schéma suivant présente ces étapes en détail et les résultats de restructuration sont affichés dans la fenêtre basse du schéma *classic Restructuration.*

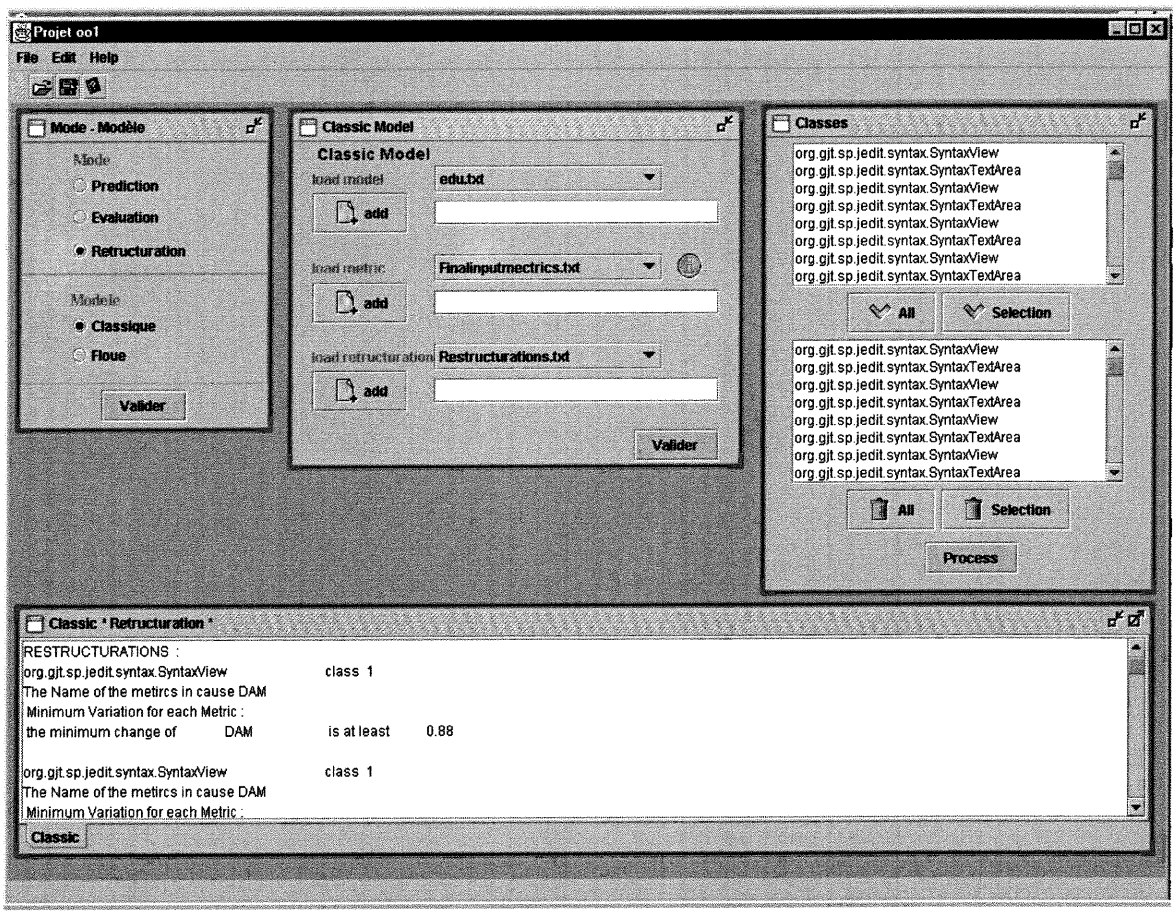


Figure 25 : Processus de restructuration d'un modèle classique

6.4.3 Conclusion et possibilités d'amélioration

Dans ce chapitre, nous avons décrit l'implantation de l'outil OO1. Ce dernier permet de prédire la stabilité des bibliothèques à objets et suggère des alternatives de restructuration afin d'améliorer la qualité des applications choisies. Cependant quelques améliorations futures seront proposées afin de rendre l'outil plus mature et plus convivial, trois types d'améliorations sont envisagés dans cette perspective:

1. Ajout d'autres fonctionnalités à l'outil.
2. Distribution et accès libre à l'outil par la proposition d'une architecture web accessible à tout le monde.
3. Amélioration de la flexibilité et de la souplesse d'utilisation (temps d'exécution, optimisation).

Pour des fins de flexibilité et de souplesse, nous prévoyons proposer des facilités d'aide et de guidance dans l'exécution des fonctionnalités de l'outil. Pour l'aspect distribution, nous avons jugé important de développer une interface web accessible à tous et exécutable sur une machine distante. Cette nouvelle architecture offrira plus de flexibilité dans l'utilisation et donne un aspect plus dynamique selon deux niveaux:

- Niveau client: par utilisation des applets Java au niveau de l'interface Web.
- Niveau serveur: par l'utilisation des Servlets pour communiquer avec le serveur.

Enfin, l'utilisation de l'outil par des experts du domaine de la qualité de logiciel leur permettra de détecter les anomalies et les conflits à corriger et proposer leurs suggestions d'amélioration.

Validation

Dans ce chapitre, nous allons valider les modèles de prédiction obtenus par extension. Nous présentons au début les objectifs de notre expérimentation. Nous décrivons la méthodologie suivie pour valider notre modèle, le déroulement de l'expérimentation et les techniques utilisées dans la validation de modèle de prédiction. Nous présentons dans la première section la validation du modèle étendu sur des échantillons de données utilisés dans la construction du modèle classique. Dans une deuxième section, nous décrivons la généralisation du modèle étendu sur des données non utilisées dans la construction de modèle de départ. Durant le processus de validation nous calculons deux métriques de performance, l'exactitude et la complétude. Enfin nous faisons une synthèse et une analyse basée sur la comparaison des résultats de prédiction obtenus par des modèles d'estimations classiques et floues.

Notons que le processus de validation des modèles de prédiction que nous envisageons de développer dans notre étude est un processus automatique basé sur l'utilisation de l'outil OO1.

7.1 INTRODUCTION

La validation est une étape primordiale dans le processus de développement des modèles de prédiction de la qualité. En effet, elle permet de s'assurer de la fiabilité, de l'exactitude et de la performance des modèles construits. Peu de travaux traitent spécifiquement de la validation des modèles de qualité. Deux type de validation sont envisageable: une validation sur des échantillons de données utilisés dans la construction du modèle de départ et une généralisation du modèle sur de nouvelles données.

La validation est vue comme une boîte noire; les données de test sont fournies au système pour avoir des classifications. Les résultats de validation sont analysés pour trouver les défaillances dans le système. L'outil de validation doit chercher les situations de disfonctionnement du système.

7.2 OBJECTIF DE L'EXPÉRIMENTATION

Afin de valider notre approche, nous cherchons a vérifier les deux hypothèses suivantes:

1. La transformation floue et l'extension préservent les relations valides du modèle initial,
2. La transformation floue et l'extension ne dégradent pas et possiblement améliorent la performance du modèle initial.

Nous avons utilisé un modèle classique qui estime la caractéristique de qualité ***stabilité***. Ce dernier est composé des règles décrites au chapitre 5. Le modèle naïf initial a été construit en utilisant un échantillon de classes choisies parmi trois systèmes. Nous avons défini la stabilité au chapitre 1 comme étant la capacité d'une classe à préserver son interface d'une version à l'autre.

Les trois principaux objectifs de la validation sont:

- Une validation du modèle sur des données qui ont été utilisées pour l'apprentissage. Nous insistons à vérifier à ce que le nouveau modèle (modèle flou) apporte une dimension supplémentaire.

- Une généralisation du modèle sur d'autres systèmes et prouver que ce dernier ne dégrade pas le degré de performance du modèle et possiblement l'améliore (validation sur des données qui n'ont pas servi pour l'apprentissage).

- Une comparaison et interprétation des résultats de validation (analyse et synthèse).

7.3 METHODOLOGIE

Nous avons conduit des expérimentations pour évaluer la performance de notre modèle de prédiction sur des données mesurées à partir de systèmes de classes. Nous étendons les modèles prédictifs existants par un ajout des connaissances de domaine et nous mesurons:

1. l'exactitude des modèles sur un ensemble de cas invisibles,
2. l'utilisation des modèles et leurs capacités explicatives.

Pour évaluer le modèle prédictif de la caractéristique **Instabilité** des classes en se basant sur les métriques de couplage, nous avons besoin des critères pour déterminer l'exactitude globale de notre modèle. L'évaluation de l'exactitude du modèle nous indique la capacité de prédiction attendue du modèle. Si le modèle basé sur les métriques d'entrées possède une bonne exactitude, cela signifie que les métriques identifient facilement les classes instables.

Deux critères d'évaluation de la prédiction sont l'exactitude et la complétude. Le tableau suivant décrit les formules de calcul de la performance d'un modèle de prédiction de l'instabilité basée sur les deux métriques qui sont l'exactitude et la complétude :

		Instabilité Prédite		
		Instable	Stable	**Complétude**
Instabilité Réelle	Instable	N_{11}	n_{12}	$\frac{n_{11}}{\sum_{i=1..2} n_{1j}}$
	Stable	N_{21}	n_{22}	$\frac{n_{22}}{\sum_{i=1..2} n_{2j}}$
	Exactitude	$\frac{n_{11}}{\sum_{i=1..2} n_{i1}}$	$\frac{n_{22}}{\sum_{i=1..2} n_{i2}}$	

Tableau 7. Matrice de classification de la performance d'un modèle de qualité

n_{11} : Le nombre de classes prédites instable qui le sont

n_{12} : Le nombre de classes prédites stables qui sont instables
n_{21} : Le nombre de classes prédites instables qui sont stables
n_{22} : Le nombre de classes prédites stables qui le sont

L'exactitude globale du modèle est calculée selon la formule suivante :

$$Exactitude = \frac{\sum_{i=1..2} n_{ii}}{\sum_{i,j=1..2} n_{ij}}$$

Pour comparer la performance du modèle obtenue par une approche floue avec celle obtenue par une approche classique, nous utilisons les mêmes données pour construire ces deux modèles. La base de règles utilisée dans le modèle flou est celle qui a été étendue par l'approche d'extension que nous avons présentée dans le chapitre 4 et 5.

Dans le but de valider la première hypothèse: "la transformation floue et l'extension préservent les relations valides du modèle initial". Nous avons tenté de comparer la performance du modèle original avec celle du modèle étendu sur des données d'apprentissage. Nous utilisons deux critères de performance: L'exactitude et la complétude. Un outil OO1 est développé pour prédire une caractéristique de la qualité en utilisant des modèles de prédiction classiques ou flous. OO1 vas servir ainsi dans l'évaluation des modèles de prédictions utilisés. Il a également la capacité de suggérer des alternatives de restructuration à fin d'améliorer la qualité d'un système évalué. Avant d'établir les modèles, nous avons conduit une première analyse pour choisir les métriques qui peuvent avoir un impact sur l'instabilité. Six d'entre eux ont été choisis.

Nous avons extrait les métriques structurelles de taille et de couplage requise à partir des versions *i* des systèmes, et comparé les classes des versions *i* et *i+1* pour déterminer la stabilité de classe. Nous avons utilisé l'outil DISCOVER pour calculer les métriques de la stabilité et l'outil OO1 pour prédire la stabilité en utilisant le modèle indiqué et la comparer à la stabilité réelle. Nos résultats prouvent que les deux modèles montrent la même exactitude (69%) et la même complétude (100%), qui nous permet de dire que notre première hypothèse est valide [32].

Pour valider la deuxième hypothèse : La transformation floue et l'extension ne dégrade pas et possiblement améliorent la performance du modèle initial. Nous avons

choisis trois bibliothèques de classes java à plusieurs versions (Beanb, Free, Jetty). Ces trois applications n'ont pas été utilisées ni dans l'apprentissage ni dans le processus de fuzzification. Elles ont été analysées par l'environnement **DISCOVER** pour calculer les métriques requise à partir des version *i* des trois systèmes, et pour comparer les classes des versions *i* et *i+1* afin de déterminer la stabilité de ces classes. Aussi bien que dans la vérification de la première hypothèse nous exploitons l'outil OO1 pour prédire la stabilité en utilisant le modèle indiqué et la comparer à la stabilité réelle.

Le schéma suivant résume ces deux types d'évaluation de la façon suivante :

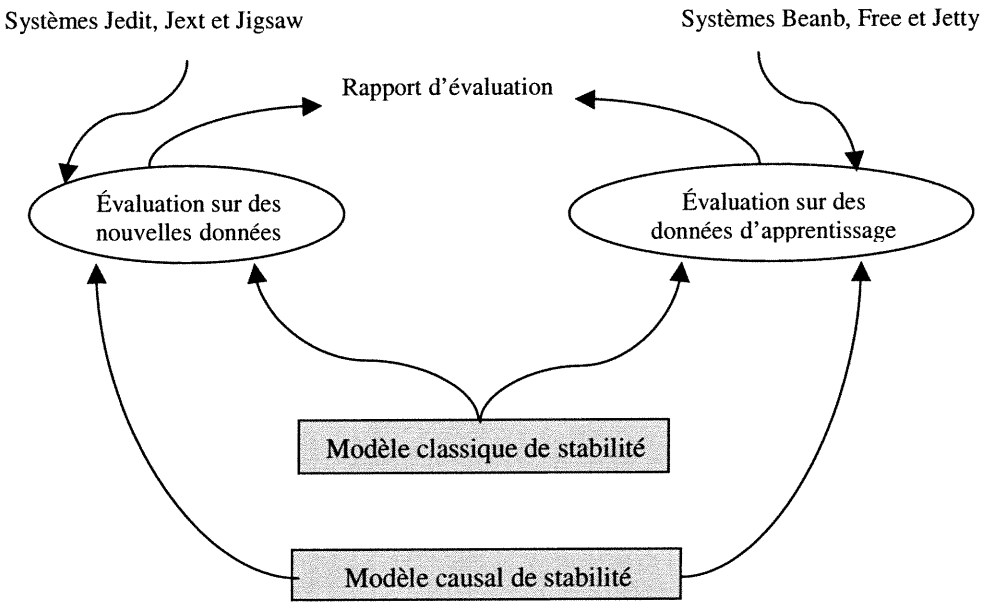


Figure 26. Évaluation des modèles de stabilité

7.4 DESCRIPTION DES DONNÉES

Nous détaillons dans cette section le déroulement de l'expérimentation et nous expliquons la collecte des données pour la validation de nos hypothèses de départ. Ensuite, nous nous attardons sur la construction des modèles de prédiction de la stabilité.

Les données des métriques utilisées dans les étapes de notre étude (l'étape de construction de modèle de prédiction, l'étape de fuzzification et l'étape de validation) sont divisées de la façon suivante, suivant les besoins de chaque étape :

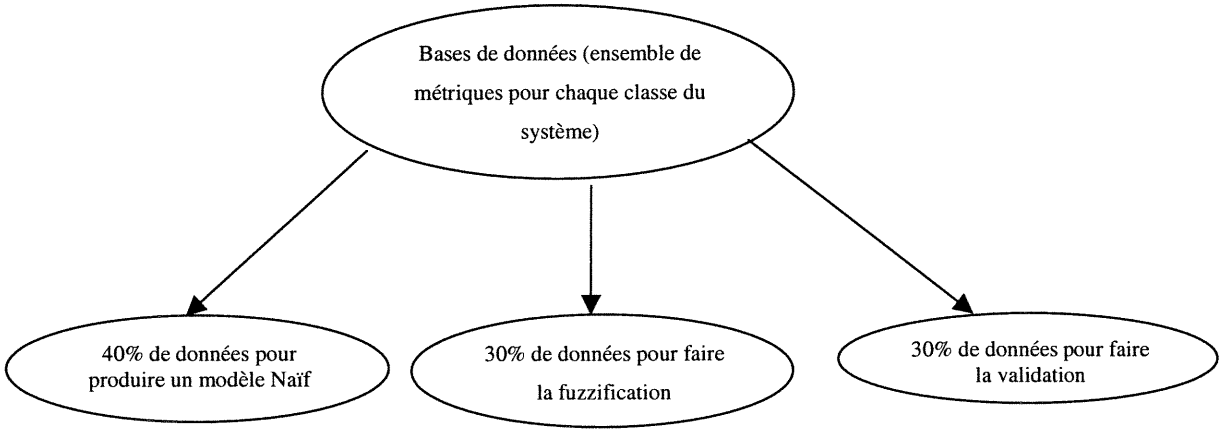


Figure 27 : Partage des données par les trois étapes de notre approche

7.4.1 Collection des données

Pour valider la première hypothèse, nous avons utilisé plusieurs versions de bibliothèques en nous limitant aux changements entre versions majeurs. Les versions sont Jedit1.2, Jedit2.6, Jext1.2, Jext2.9, Jigsaw1.02 et Jigsaw2.1.2. Nous avons fusionné les valeurs des métriques des trois applications afin d'utiliser un nombre de classes significatif. Le nombre total de classes utilisé est 486 classes dont 336 sont instables et 150 sont stables entre deux versions majeures des systèmes. Ces derniers ont été utilisés dans d'apprentissage.

Applications	Version	Nombre de Classes	Nombre de Classe Stable	Nombre de Classe Instable
Jedit	1.2 à 2.6	82	2	80
Jext	1.2 à 2.9	48	3	45
Jigsaw	1.0.2 à 2.1.2	356	145	211
Total		486	150	336

Tableau 8. Description des trois systèmes utilisés dans la construction des modèles de départ.

Le graphe suivant présente schématiquement le nombre de classe stables et instables pour chaque système utilisé dans la construction du modèle naïf.

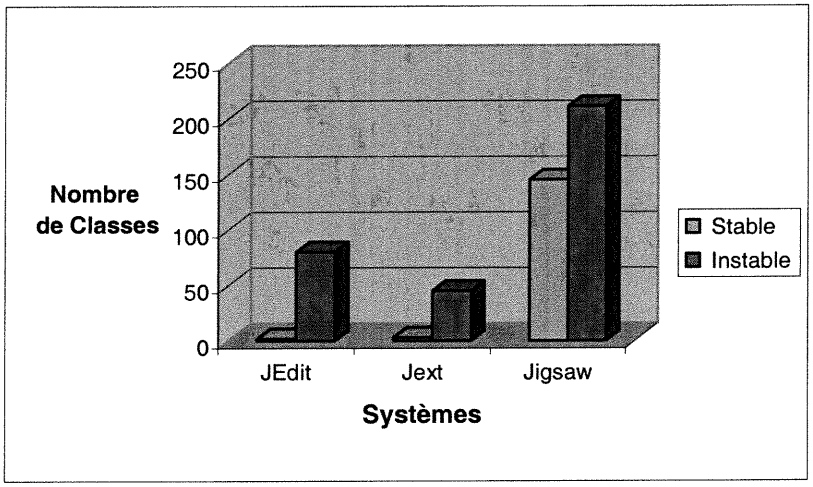


Figure 28 Description des systèmes utilisés dans l'apprentissage

Pour la validation de la deuxième hypothèse, Nous avons choisis les trois bibliothèques de classes Beanb, Free et Jetty. Ces nouveaux systèmes n'ont pas été servis dans l'apprentissage. Le tableau suivant décrit le nombre de classes stables et instables pour chacune de ces applications.

Application	Version	Nombre de classes	Nombre de classes stables	Nombre de classes instables
Beanb	9.2 à 9.6	391	97	294
Free	100 à 16	50	27	23
Jetty	2.4.9 à 3.1.5	229	173	56

Tableau 9. Description des trois systèmes non utilisés dans l'apprentissage

Le graphe suivant présente schématiquement le nombre de classe stable et instable pour chaque système utilisé dans le processus de généralisation.

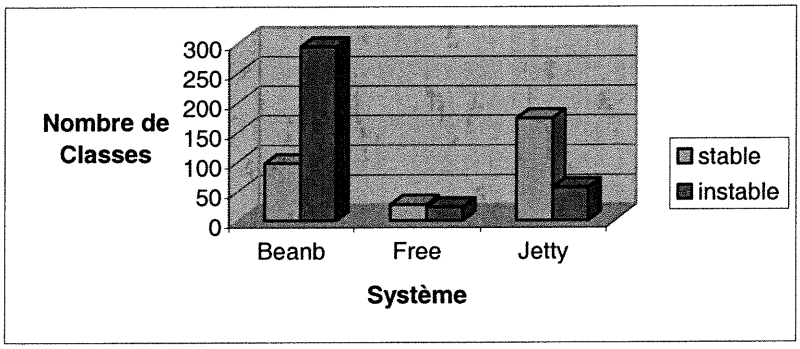


Figure 29 Description de nouveaux systèmes utilisés dans la généralisation du modèle de prédiction

7.5 RÉSULTATS ET ANALYSE

Nous présentons au début de cette section les résultats de validation obtenus par les techniques décrites plus haut. Ces résultats prennent la forme d'une matrice de classification de la performance pour les modèles classique et flou étendu .

7.5.1 Résultats sur les systèmes d'apprentissage.

Nous utilisons deux modèles de prédiction A1 et A2 dont la description est présentée dans les chapitres 4 et 5.

Le modèle A1: Le modèle flou étendu
Le modèle A2: Le modèle classique de départ

7.5.1.1 Calcul de l'exactitude pour le modèle flou

Instabilité Prédite

Instabilité Réelle		Instable	Stable	Complétude
	Instable	**336**	**0**	**1**
	Stable	**150**	**0**	**0**
	Exactitude	**0.69**	**0**	

Tableau 10. Matrice de classification de la performance du modèle A1

Exactitude globale = **0.69**

7.5.1.2 Calcul de l'exactitude pour le modèle classique

Instabilité Prédite

Instabilité Réelle		Instable	Stable	Complétude
	Instable	**336**	**0**	**1**
	Stable	**150**	**0**	**0**
	Exactitude	**0.69**	**0**	

Tableau 11. Matrice de classification de la performance du modèle A2

Exactitude globale = **0.69**

7.5.2 Résultats sur les nouveaux systèmes.

Nous allons présenter dans cette section les résultats d'évaluation des modèles sur de nouvelles données non exploitées dans l'apprentissage. Nous calculons la performance (exactitude, complétude) de chaque modèle (classique, floue) pour les trois systèmes choisis.

Système Beanb

- Calcul de l'exactitude pour le modèle flou

Instabilité Prédite

Instabilité Réelle		Instable	Stable	Complétude
	Instable	**278**	**17**	**0.94**
	Stable	**65**	**31**	**0.32**
	Exactitude	**0.81**	**0.65**	

Tableau 12. Matrice de classification de la performance du modèle A1

Exactitude globale = **0.79**

- Calcul de l'exactitude pour le modèle classique

Instabilité Prédite

Instabilité Réelle		Instable	Stable	Complétude
	Instable	**292**	**0**	**1**
	Stable	**99**	**0**	**0**
	Exactitude	**0.75**	**0**	

Tableau 13. Matrice de classification de la performance du modèle A2

Exactitude globale = **0.74**

Système Free

- Calcul de l'exactitude pour le modèle flou

Instabilité Prédite

Instabilité Réelle	Instable	Stable	Complétude
Instable	**14**	**9**	**0.61**
Stable	**9**	**18**	**0.67**
Exactitude	**0.61**	**0.67**	

Tableau 14. Matrice de classification de la performance du modèle A1

Exactitude globale = **0.64**

- Calcul de l'exactitude pour le modèle classique

Instabilité Prédite

Instabilité Réelle	Instable	Stable	Complétude
Instable	**21**	**0**	**1**
Stable	**29**	**0**	**0**
Exactitude	**0.42**	**0**	

Tableau 15. Matrice de classification de la performance du modèle A2

Exactitude globale = **0.46**

Système Jetty

- Calcul de l'exactitude pour le modèle flou

Instabilité Prédite

Instabilité Réelle	Instable	Stable	Complétude
Instable	**29**	**27**	**0.52**
Stable	**111**	**62**	**0.36**
Exactitude	**0.21**	**0.7**	

Tableau 16. Matrice de classification de la performance du modèle A1

Exactitude globale = **0.39**

- Calcul de l'exactitude pour le modèle classique

		Instabilité Prédite		
		Instable	Stable	Complétude
Instabilité Réelle	Instable	**48**	**0**	**1**
	Stable	**181**	**0**	**0**
	Exactitude	**0.21**	**0**	

Tableau 17. Matrice de classification de la performance du modèle A2

Exactitude globale = **0.2**

7.5.3 Analyse des résultats

Le modèle flou est toujours fidèle aux concepts des modèles classiques tout en étant plus général car incluant les connaissances du domaine pour expliquer les tendances entre les variables du modèle.

Les résultats de validation obtenus sur les systèmes d'apprentissage prouvent que les deux modèles offrent la même exactitude (69%) et la même complétude (100%), ce qui nous permet de dire que notre première hypothèse est valide [32].

En ce qui concerne la validation sur les nouveaux systèmes, la valeur de la complétude du modèle initial donne 100% pour les trois systèmes. Le modèle étendu donne respectivement 94%., 61% et 52% et La non concordance entre les résultats des deux modèles est facilement interprétée par l'existence d'une règle de classification par défaut dans le modèle initial (modèle classique); celle-ci prédit que la classe est instable si aucune règle n'est appliquée. Durant le processus d'extension, nous avons décidé d'éliminer cette règle de la base de règles [32].

Le graphe suivant résume les résultats de validation obtenus pour la mesure d'exactitude des nouveaux systèmes évalués sur un modèle classique et flou.

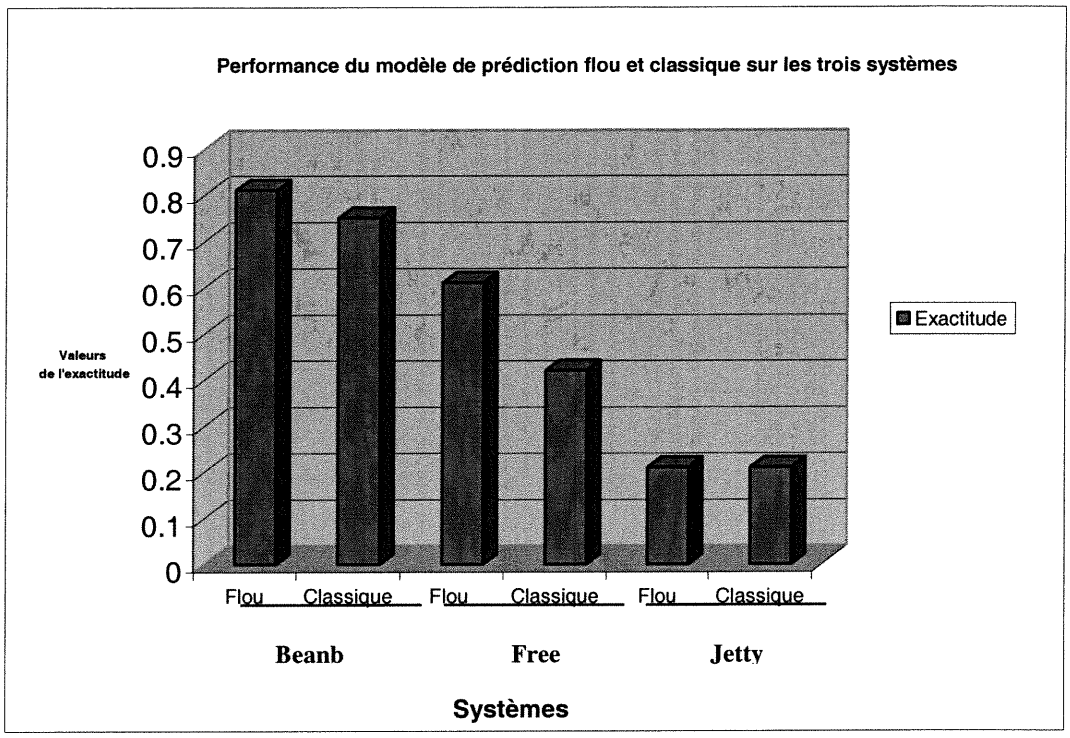


Figure 30. Exactitude des trois systèmes Beanb, Free et Jetty

En comparant les deux approches floue et classique en terme d'exactitude, les résultats sont très intéressants. Pour les systèmes **Free** et **Beanb**, le modèle étendu améliore d'une manière considérable l'exactitude de la prédiction. Pour le système **Free**, il passe de 42% à 61%, pour le système **Beanb**, il passe de 75% à 81% et pour le système **Jetty**, les deux modèles donnent la même complétude. Donc, la deuxième hypothèse est confirmée [32]. Quant à la complétude de prédiction, le biais introduit par la règle de classification par défaut dans le modèle classique ne permet pas d'écrire la conclusion.

Le graphe suivant englobe les résultats obtenus sous forme d'exactitude de prédiction de l'instabilité des classes pour les systèmes d'apprentissage et pour les nouveaux systèmes (Beanb, Free et Jetty) sur les deux modèles classique et flou.

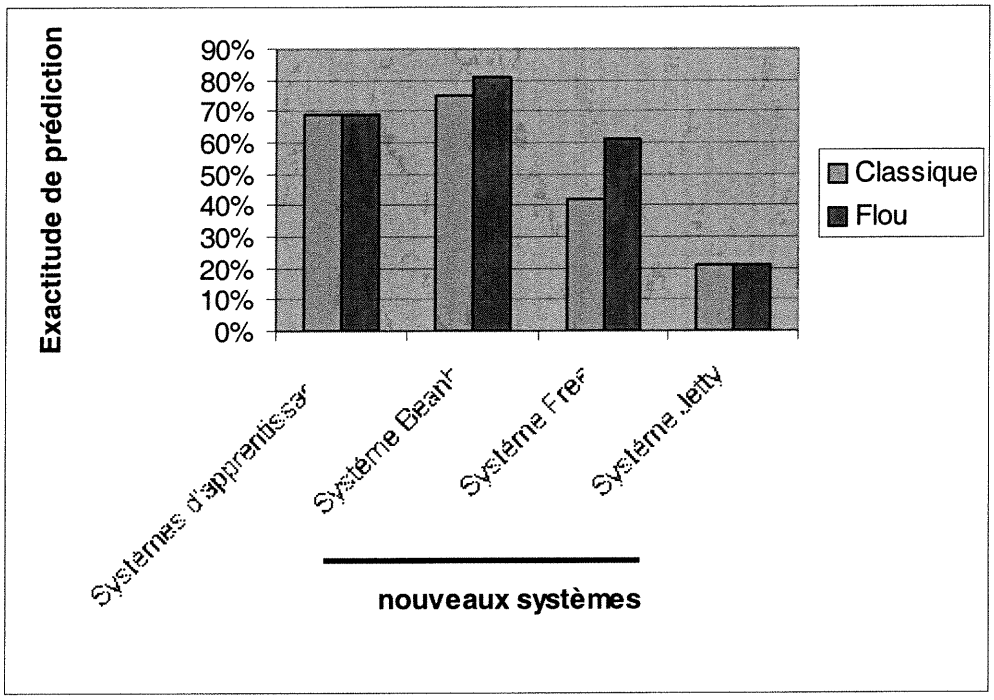


Figure 31 Description de l'exactitude de prédiction pour les deux modes d'évaluation

L'ensemble des résultats obtenus par notre approche montre que les modèles flous étendus peuvent remplacer les modèles naïfs tout en améliorant l'exactitude de prédiction de la qualité. En effet, l'analyse des résultats révèle que:

- la fuzzification du modèle classique préserve la performance du modèle original.
- le modèle étendu, d'une part rend le modèle plus utile dans la prise de décision, d'autre part peut réellement améliorer les résultats de prédiction. Ceci peut être expliqué par les nouvelles métriques qui sont introduites lors du processus d'extension.
- les modèles flous étendus offrent systématiquement une meilleure performance que les modèles classiques.

Conclusion

Le but de notre projet est de proposer une approche de transformation floue et d'extension d'un modèle naïf pour l'étude de la stabilité des applications orientés objet.

Nous avons présenté un cadre d'application pour développer et utiliser les modèles prédictifs de qualité de logiciel basés sur les règles. En utilisant des outils basés sur la logique floue, nous avons pu éviter le problème des valeurs seuils dans les modèles classiques. En outre, en étendant nos modèles avec les heuristiques du domaine, nous avons présenté la solution à l'établissement d'un lien sémantique entre les attributs internes et externes de logiciel, rendant les modèles plus utiles en tant qu'outils de support de décision.

En vue d'implémenter notre approche, nous avons développé un outil appelé OO1 qui permet de prédire une caractéristique de qualité, d'évaluer un modèle de qualité sur un échantillon de données et de suggérer des alternatives de restructuration. Nous avons appliqué cet outil pour étudier la stabilité de plusieurs systèmes et nous avons obtenus des résultats probants. Nous avons aussi bénéficié de l'utilisation de l'environnement DISCOVER pour analyser des bibliothèques à objets et extraire les métriques de taille, d'héritage et de couplage requises pour estimer la caractéristique stabilité.

Nous avons déduit à travers le processus de validation que le modèle flou étendu préserve la performance du modèle initial et possiblement l'améliore. Cela peut être expliqué par la présence de relations sémantiquement plus riches dans le modèle étendu et par l'ajout de nouvelles métriques durant le processus de prédiction.

Enfin, nous pouvons affirmer que notre modèle peut facilement prédire une caractéristique de qualité selon un mode classique ou flou. Les résultats obtenus avec un modèle de prédiction obtenu par extension sont plus intéressants de part la souplesse de raisonnement, les possibilités de raffinement et une programmation plus instinctive offertes par la logique floue. Dans tous les cas, l'ajout des heuristiques du

domaine enrichissent les relations de causalité par des informations supplémentaire reflétant la réalité d'une façon adéquate on grandement amélioré la pertinence de notre modèle comme support de décision.

Les hypothèses exprimées au début de notre étude ont été vérifiées dans l'étape de validation de l'approche. Il reste à généraliser l'approche présentée dans ce mémoire afin de renforcer la fiabilité de nos résultats.

L'ensemble des travaux présentés dans ce mémoire apporte une contribution substantielle à l'étude sur la stabilité des bibliothèques de classe à objets. Bien que les résultats obtenus soient encourageants, il reste des issues ouvertes pour amélioration. Des avenues de recherche peuvent s'ouvrir comme suite logique à ce travail. Nous identifions, en particulier les directions suivantes:

- Développement d'une technique permettant l'automatisation du processus d'ajout des heuristiques du domaine en créant un dépôt des heuristiques et en définissant des algorithmes efficaces pour exploiter son contenu (problème de recherche de chemin et donc possiblement prendre en compte les problème de convergence).
- Application à d'autres modèles
- Application à d'autres caractéristiques
- Développement d'un algorithme automatique pour générer les règles floues.
- Exploitation de la technique Neuro-Floue qui offre le potentiel de mettre en œuvre un processus automatique de conversion d'un modèle naïf en model causal .

Bibliographies

[1] Roger S. Pressman, *Software Engineering a practitioner's approach.* McGraw-Hill, 4ème édition, 1997.

[2] V. R. Basili, L. Briand & W. Melo, *How Reuse Influences Productivity in Object-Oriented Systems*. Communication of the ACM, Vol30, N. 10, app104-114, 1996.

[3] S. Demeyer, S. Ducasse, *Metrics, Do they really help ?*. Langages et Modèles à Objet, 1999.

[4] M. W. Price and S. A Demurjian, *Analyzing and measuring Reusability in Object-Oriented Design*. In Proc. Of OOPSLA'97, 1997.

[5] W Li, S. Henry, *Object-Oriented Metrics that Predict Maintainability.* journal of systems and Software, 23(2), 111-122, 1993.

[6] Y.Mao, H.A.Sahraoui & H.Lounis *Reusability Hypothesis Verification Using Machine Learning : A Case Study.* In Procof the 13th IEEE International Automated Software Engineering Conference, p. 84-93, Honolulu, Hawai, October 13-16, 1998.

[7] Briand L.C, WustJ., IkonomovskiS., Lounis H. *Investigating Quality Factors in Object-Oriented Designs : an Industrial Case Study.* In the 21st International Conf on Software Engineering(ICSE), Los Angeles, USA, May 16-22, 1999.

[8] M.A De Almeidia, H. Lounis & W. Melo. *An Investigation on the Use of Machine Learned Models for Estimating Software Correctability.* In the international Journal of Software Engineering and Knowledge Engineering, P. 565-593, vol. 9, number 5, 1999.

[9] Fenton NE, *Software Metrics a rigorous approach.* Chapman & Hall, edition, 1992.

[10] Fenton NE and Neil M, *Software Metrics and Risk.* Proc 2nd European Software Measurement Conference (FESMA'99),TI-KVIV, Amsterdam, ISBN 90-76019-07-X39-55, 1999.

[11] Fenton NE and Neil M, *Software Metrics: Roadmap.* In Proc. Of the 22nd International Conference on Software Engineering, 2000.

[12] Page principale de l'outils commercial HUGIN. The HUGIN Decision Engine http://www.hugin.com/products/software/BNengine/

[13] Page sur l'outil d'Apprentissage automatique C4.5 (téléchargement gratuit) http://www.cs.nyu.edu/~binli/pc4.5/

[14] Houari A.Sahraoui, Mounir Boukadoum, Hakim Lounis. *Building Quality Estimation models with Fuzzy Threshold Values*. l'Objet, Vol. 17, No. 4, Edition Hermès Sciences, 2001.

[15] Wittig, G., *Estimating Software Development Effort with Connectionist Models Working*. Paper Series 33/95, Monash University, 1995.

[16] Kumar, S., Krishna, B.A., and Satsangi, P.S, *Fuzzy Systems and Neural Networks in Software Engineering Project Managemen.* J. Applied Intelligence 4,31-52, 1994.

[17] Bastani, F.B., Dimacro, G., and Pasquini, A., *Experimental Evaluation of a Fuzzy-Set Based measure of Software Correctness Using Program Mutation.* In Proc. 15th Int. Conf. Software Engineering, 45-54, 1993.

[18] Munakata, T., and Jani, Y., *Fuzzy Systems : An Overview*. Comm. ACM 37,69-76, 1994.

[19] Jang, R.J.-S., *ANFIS : Adaptive_Network-Based Fuzzy Inference*. System, IEEE Trans. Systems, Man, and Cybernetics 23, 665-685, 1993.

[20] Horikawa, S., Furnuhashi,T., and Ucikawa, Y., *On Fuzzy Modelling Using Fuzzy Neural Networks with the Back-Propagation Algorithm.* IEEE Trans. Neural Networks 3, 801-806, 1992.

[21] Aha,D.W, *Case_Based learning Algorithms, in Proceedings of the DARPA Case_Based Reasoning*. Workshop, Mogan Kaufmann, Washington, D.C.,147 158, 1991.

[22] Mukhopadhyay,T.,Vicinan Za,S.S,and Prietula,M.J, *Examining the Feasibility of a Case Based Reasoning Model for Software Effort Estimation*. MIS Quarterly 16,155-171, 1992.

[23] Andrew R. Gray, Stephen G. MacDonell *A Comparaison of Techniques for Developing Predictive Models of Software Metrics*. Computer and information Science University of Otago, Dunedin, New Zealand.

[24] L. A. Zadeh, *Probability measures of fuzzy events*. Journal Math. Anal. Applic., 23. reprinted in Fuzzy Sets and Applications: selected papers by L. A.Zadeh, pp. 45-51, 1968.

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

[26] J.R. Quinlan, C4.5: *Programs for Machine Learning.* Morgan Kaufmann Pub., 1993.

[27] C. Marsala, and B. Bouchon-Meunier. *Fuzzy partioning using mathematical morphology in a learning scheme.* Proc. of 5th Conference on Fuzzy Systems, 1996.

[28] J.J. DeGruijter and A.B. McBratney, *A modified fuzzy k means for predictive classification.* In: Bock,H.H.(ed) Classification and Related Methods of Data Analysis. pp. 97-104. Elsevier Science, Amsterdam, 1988.

[29] J.C. Bezdek, R. Ehrlich and W. Full. *FCM: The Fuzzy c-Means Clustering Algorithm.* Computers and Geoscience, 10, pp. 191 – 203, 1984.

[30] Houari A. Sahraoui, Mohamed. Adel Serhani, Mounir Boukadoum *Extending Software Quality Predictive Models Using Domain Knowledge.* In 5th International ECOOP Workshop on Qantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2001) Budapest, Hungary June, 2001

[31] Houari A. Sahraoui, Mounir Boukadoum, Mohamed Serhani, Hassan M. Chawiche, Gang Mai *Building and Using Rule-Based Software Quality Prediction Models.* (Submited to ICSE).

[32] Houari Sahraoui, Mounir Boukadoum and Mohamed Adel Serhani *From Naïve to Causal Software Quality Prediction Models.* Technical report DIRO Université de Montréal

[33] E. H. Mamdani and B.R. Gaines, (ed). *Fuzzy Reasoning and its Applications.* Academic Press.

[34] Hassan Chawiche *OO1_Correct : Un Environnement De Restructuration Des Programmes Objets Piloté Par La Qualité.* Thèse en cours département d'informatique et de recherche opérationnelle. Mai, 2002.

Annexe

EVALUTION REPORT

Contents

1/-System Description:

Model Name : Stability_Model

Model View:

Name of metrics Implicated :
DIT , OCAEC, DAM, NOP, NAM, NAA

2/-Evaluation Data

Metrics	OCAEC	NOP	DIT	NAM	NAA	DAM
Averages	0.09	0.11	1.57	13.18	3.99	0.5
median	0.0	0.0	1.0	7.0	2.0	0.57
mode	0.0	0.0	1.0	2.0	0.0	0.0
Variance	0.2	1.36	2.22	263.9	289.86	290.08
Derivation	0.45	1.17	1.49	16.24	17.03	17.03

More Details And Graphs

3/-Prediction Result:

Number Of Cases in Class 1:343.0

Number Of Cases in Class 2:48.0

Predicted Unstability

Real Unstability		Unstable	Stable	Completeness
	Unstable	278.0	65.0	0.81
	Stable	17.0	31.0	0.65
	Exactness	0.94	0.32	

4/-Restructuration Result:

Exemple :

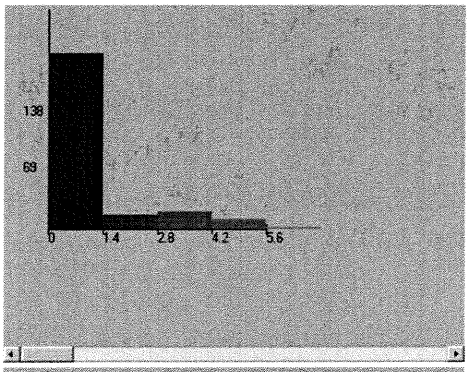
Class javax.servlet.jsp.PageContext :

The evaluation of the quality of this class indicates that it has a Stability SMALL , with a degre of truth equal to 1.0
Because it is a negative classification, so it is NECESSARY to decrease this truth value,
to enhance the classification of this class.
To have this improvement, you should change the values of metrics which are in cause of this unwanted classification.
The metrics in cause are:
- Data access metric- (DAM)
>>> DAM : You have to Increase its value, between 0.17 and 0.26

Suggestion 1 : we suggest the transformation that converting an association modeled using inheritance into an aggregation
Within the context of this restructuration, the class javax.servlet.jsp.PageContext will be the initial subclass,
then, the restructuration able to Increase the crisp value of Data access metric, between *--QuDAAtr+QuPAtr--* and *--QuDAAtr+QuPAtr--*.
where QuDAAtr is the quotient of (DAMxNumIATR) per (NTA-NumIATR) ,and QuPAtr is the quotient of NPA per ((NTA+1)xNTA).
> In QuDAAtr : NumIATR is the number of attributes inherited ,and NTA is the number of total attribut.
> In QuPAtr : NPA is the number of public attribut ,and NTA is the number of total attribut.

5/-More Details and Graphs:

Graphs And Details



Metric Name: **DIT**

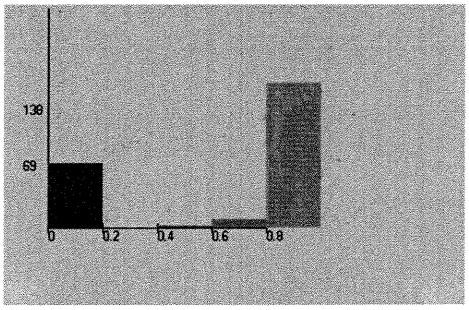
Average: 1.090000033378601

Median : 1.0

Mode: 0.0

Variance: 2.170538321106966

Derivation : 1.4732746930246803



Metric Name: **DAM**

Average: 0.6899999976158142

Median : 1.0

Mode: 1.0

Variance: 2.369785401383496

Derivation : 1.5394107318657668