UNIVERSITÉ DU QUÉBEC

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

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
NICHOLAS DROUIN

ÉVOLUTION DES SYSTÈMES ORIENTÉS OBJET :
COMPRÉHENSION DE L'ÉVOLUTION DE CERTAINS ASPECTS RELATIFS À LEUR QUALITÉ EN UTILISANT LES MÉTRIQUES

Octobre 2012

À ma mère

À la mémoire de mon père

ÉVOLUTION DES SYSTÈMES ORIENTÉS OBJET : *COMPRÉHENSION DE L'ÉVOLUTION DE CERTAINS ASPECTS RELATIFS À LEUR QUALITÉ EN UTILISANT LES MÉTRIQUES*

Nicholas DROUIN

SOMMAIRE

La programmation orientée objet a été largement utilisée dans les milieux industriels. Les systèmes développés sont dans la plupart des cas complexes et de grande taille. Par ailleurs, leur évolution est souvent inéluctable. Elle présente des enjeux économiques et qualitatifs importants. Ces systèmes subissent, en effet, beaucoup de changements qui peuvent avec le temps altérer leur qualité. Dans certains cas, des actions de restructuration s'imposent. Dans ce contexte, les métriques logicielles constituent des outils importants. Elles peuvent être utilisées pour évaluer (comprendre, gérer et contrôler) l'évolution de certains aspects relatifs à leur qualité.

Nous avons utilisé un modèle basé sur le flux de contrôle (en particulier les interactions entre classes) et les probabilités (parcours du flot de contrôle) pour les systèmes orientés objet (Badri et al., Journal of Object Technology, 2009). Il tient compte de façon indirecte des dépendances de contrôle entre classes. Ces dépendances peuvent être très complexes. Lors de l'évolution des systèmes, elles peuvent avoir un impact important sur différentes caractéristiques de leur qualité (entre autres). L'objectif principal du modèle étant de pouvoir capturer, de façon intégrée, de l'information relative à différents attributs logiciel.

Nous nous intéressons, dans ce mémoire, à l'évaluation des capacités du modèle à suivre (refléter) plusieurs facettes relatives à l'évolution de systèmes logiciels Java réels (évolution de certains aspects relatifs à la qualité, amélioration vs dégradation, modifications, etc.). Il s'agira d'expérimenter le modèle sur différents systèmes réels (évolution à travers plusieurs versions) et de le comparer à plusieurs métriques orientées objet existantes (reflétant plusieurs aspects relatifs à leur qualité).

EVOLUTION OF OBJECT ORIENTED SYSTEMS: *UNDERSTANDING THE EVOLUTION OF SOME ASPECTS RELATED TO SOFTWARE QUALITY USING METRICS*

Nicholas DROUIN

ABSTRACT

Object oriented programming has been largely used in industrial environments. Developed systems are, for most of the cases, complex and large. Moreover, their evolution is often inevitable. It presents important economical and qualitative issues. These systems experience, in fact, many changes over time that may affect their quality. In some cases, refactoring activities are needed. In this context, software metrics are important tools. They can be used for evaluating (understanding, managing, controlling) the evolution of some aspects related to software quality.

We used a model based on control flow (especially the interactions between classes) and probabilities (control flow path) for object-oriented systems (Badri et al., Journal of Object Technology, 2009). It takes into account indirect control dependencies between classes. Such dependencies can be very complex. When systems evolve, they can have a significant impact on various characteristics of their quality (among others). The main purpose of the model is to be able to capture, in an integrated way, information on various software attributes.

In this project, we are interested in the assessment of the capacities of the model to reflect several aspects relative to the evolution of real Java software systems (evolution of certain aspects of their quality, improvement vs. degradation, changes, etc.). We experiment the model on different real systems (evolution through several versions) and compare it to various existing object-oriented metrics (reflecting many aspects of software quality).

REMERCIEMENTS

Merci,

À mon directeur de recherche, Mourad Badri, pour m'avoir guidé tout au long de ma recherche. C'est un honneur pour moi d'avoir été dirigé par vous dans mes travaux. Merci pour les encouragements, la disponibilité et le soutien à la fois moral et financier dont vous m'avez fait part tout au long de ces années à la maîtrise. Merci aussi pour votre confiance à mon égard depuis avant-même ma maîtrise.

À Fadel Touré pour ton aide précieuse tout au long de mes recherches. Merci pour tes conseils et tes recommandations tout au long de ma maîtrise.

À mes parents qui m'ont toujours encouragé à aller le plus loin possible dans mes études. Merci à ma mère pour ton soutien continu tout au long de mes études et merci à mon père pour me guider de tout là-haut. Ce mémoire vous est dédié.

À ma famille et aux amis qui m'ont aidé et encouragé à la poursuite de mes études. Certains moments furent plus difficiles que d'autres, et vous m'avez permis de faire le vide dans ces moments ardus.

Au Fonds québécois de recherche en Nature et technologie (FQRNT) pour la bourse que vous m'avez accordée.

À François Meunier et Fathallah Nouboud, évaluateurs de ce mémoire, pour vos précieux commentaires et pour le temps que vous aurez pris à la lecture de ce mémoire.

Enfin, un gros merci à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce mémoire. Ce fut grandement apprécié et c'est grâce à vous que je suis parvenu à persévérer jusqu'au bout. Merci !

TABLE DES MATIERES

SECTION 3 : LES EXPÉRIMENTATIONS

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE 1
INTRODUCTION

La programmation orientée objet est un paradigme très présent dans les milieux industriels. Les systèmes logiciels développés dans ce contexte sont souvent d'une taille importante et d'une complexité qui l'est tout autant. Il s'agit souvent d'investissements majeurs. Par conséquent, ces systèmes sont utilisés et maintenus sur une très longue période de temps.

La raison d'être d'un système logiciel est de combler certains besoins existants dans le monde réel. La parcelle de réalité que tente de représenter le système logiciel est toutefois en perpétuel changement. Par conséquent, un système doit être adapté de façon continue pour réussir à survivre à ces changements (Ali, 2009). La découverte de fautes est un autre motif justifiant les modifications apportées à un système (Sommerville, 2007). En effet, les problèmes latents doivent être corrigés pour que le système soit pleinement utilisable. Bref, l'évolution d'un logiciel est inévitable.

Essentiellement, apporter des changements à un système logiciel est fait dans un objectif d'amélioration. Toutefois, faire évoluer un système logiciel peut avoir d'importantes conséquences imprévisibles et non désirées. Des modifications apportées aux parties de code plus critiques peuvent, par un effet domino, affecter la qualité d'autres parties du système et ainsi, entraîner l'apparition de nouveaux problèmes (Parnas, 1994; Lehman, 1997; Van Gurp, 2002; Zhang, 2010). On parle alors d'une détérioration de la qualité du système. De tels effets peuvent donc avoir des conséquences qualitatives et économiques importantes (Parnas, 1994; Van Gurp, 2002). De plus, la demande pour des systèmes logiciels de qualité est en forte croissance (Aggarwal, 2007). Un contrôle continu et assidu de la qualité au cours de l'évolution d'un système est par conséquent crucial.

La qualité logicielle est toutefois une notion très complexe et globale. Elle se présente sous diverses facettes (résistance aux fautes, maintenabilité, etc.)

(Aggarwal, 2007). Dans ce contexte, les métriques peuvent s'avérer d'une grande aide. Elles fournissent des informations importantes sur certains aspects de la qualité, et ce, par la capture d'attributs internes des systèmes. Elles sont des outils importants pour l'évaluation (compréhension, gestion, contrôle) de certaines dimensions de la qualité (Aggarwal, 2007; Badri, 2009; Briand, 2000; Subramanyam, 2003; Zhou, 2006; Dagpinar, 2003).

Problématique

L'efficacité des métriques orientées objet dans l'évaluation (capture) de certaines dimensions de la qualité d'un système logiciel a été maintes fois validée empiriquement dans la littérature (Basili, 1996; Subramanyam, 2003; Dagpinar, 2003; Zhou, 2006; Lee, 2007; Murgia, 2009; Briand, 2000; Singh, 2010; Badri, 2011; Eski, 2011). Les métriques, mesurées pour la plupart sur les classes d'un système, permettent par exemple de distinguer assez efficacement certaines classes plus sujettes aux fautes ou à des modifications (Aggarwal, 2007; Briand, 2000; Reddy, 2009; Subramanyam, 2003; Zhang, 2009 ; Zhou, 2006). En se plaçant dans le contexte d'un système en perpétuel changement, les capacités des métriques quant à l'évaluation des attributs logiciels deviennent soudainement beaucoup plus complexes à interpréter. Il n'est alors plus question d'une unique capture de système mise à l'étude, mais plutôt d'une étude sur un ensemble de versions distancées temporellement constituant plusieurs captures de l'évolution continue du système au cours du temps.

Le présent mémoire se place dans le contexte particulier de l'évolution logicielle. Plus précisément, nous nous intéressons à l'évolution de leur qualité. Notre objectif est de déterminer dans quelle mesure des métriques orientées objet reconnues pour leur pouvoir prédicatif de la qualité peuvent aider à mieux comprendre comment la qualité se comporte (évolue) dans le contexte d'un système logiciel en évolution. Nous allons aussi, dans ce contexte, nous intéresser à une métrique élaborée par Badri et al. (Badri, 2009), les indicateurs de qualité (Qi). Ce modèle a déjà fait l'objet d'importantes études empiriques qui ont démontré ses capacités à unifier de

nombreux et importants attributs logiciels (Badri, 2009), tels que la taille, la complexité et le couplage. Ce modèle tient compte de plusieurs dimensions reliées à la qualité (Touré, 2007). Nous pensons que ce modèle peut également être utilisé pour mieux comprendre, gérer et contrôler la qualité dans un système en continuel changement.

Approche

Le présent mémoire s'inscrit donc dans cet ordre d'idées. Pour parvenir à valider notre hypothèse principale, nous avons procédé à une évaluation empirique d'envergure intégrant plusieurs expérimentations. Celles-ci ont principalement pour objectif d'apporter des éléments de réponse à la problématique globale. C'est-à-dire, nous voulons valider la pertinence d'utiliser les métriques pour aider à la compréhension de l'évolution de certains aspects relatifs aux systèmes orientés objet. Trois systèmes ont été analysés sur une longue période, au travers d'une importante quantité de captures, afin d'analyser l'évolution de plusieurs attributs de leur qualité. Disposer de ces données nous permet d'analyser la qualité sous plusieurs facettes. En particulier, nous nous intéressons à la qualité selon les perspectives suivantes.

- En premier lieu, la qualité est analysée selon une perspective interne du système. Nous considérons, dans ce contexte, les métriques orientées objet (du moins celles que nous avons sélectionnées) comme des indicateurs de référence au niveau système. Nous avons pour objectif d'étudier comment les indicateurs de qualité (Qi), unifiant plusieurs attributs logiciels, se comportent comparativement aux métriques de référence. L'étude est effectuée à plusieurs niveaux. Tout d'abord, par une analyse des relations entre les métriques au niveau système, en procédant à une étude de courbes et des corrélations entre les Qi et les métriques orientées objet. L'idée étant de voir si les Qi capturent l'information produite par les métriques orientées objet. Ensuite, par une analyse sur l'influence de la croissance en taille du système sur les valeurs moyennes des métriques.

- En deuxième lieu, et toujours dans le même contexte, la qualité est analysée selon une perspective externe du système. Nous considérons cette fois-ci comme indicateur de référence de la qualité le nombre de fautes répertoriées pour chaque version d'un système durant son évolution. Notre objectif étant de voir dans quelle mesure les métriques (incluant les Qi) arrivent à capturer l'évolution de la qualité sous cette perspective. Nous considérons aussi la fréquence de changements comme métrique potentielle. Une étude de corrélations sera effectuée pour vérifier nos hypothèses.

- En troisième lieu, nous allons nous intéresser à l'utilisation des métriques pour illustrer les diverses lois de Lehman (complexité croissante, changement continu, etc.) sur l'évolution. Nous allons nous concentrer sur certaines lois en lien avec les attributs de qualité. Nous nous intéressons particulièrement aux performances du modèle Qi relativement aux différentes métriques orientées objet reconnues pour capturer différents attributs de la qualité.

Organisation du mémoire

Ce mémoire comporte trois parties majeures. La première de ces parties présente un état de l'art étendu sur les dynamiques d'évolution et la qualité logicielle. Le but étant de synthétiser l'essentiel des connaissances nécessaires pour la compréhension de la problématique. L'utilisation des métriques est mise à l'avant pour ces deux contextes.

La deuxième partie présente le modèle des Qi tel que défini dans (Badri, 2009; Badri, 2012). L'accent est mis sur la définition du modèle et sur les études empiriques effectuées démontrant certaines de ces capacités.

La troisième partie du mémoire présente les différentes étapes (séries d'expérimentations) de l'étude empirique d'envergure que nous avons effectuée pour tenter de répondre aux différentes facettes de notre problématique. Nous présentons, tout d'abord dans cette partie, les lignes directrices de nos expérimentations, la

sélection des métriques ainsi que les processus de cueillette de données. Les résultats sont ensuite présentés sous la forme de trois (séries d') expérimentations distinctes, structurées de la façon suivante :

- Une étude abordant la qualité selon une perspective interne au système, se basant sur des attributs relatifs à sa structure. Nous divisons cette section en trois parties. Premièrement, une étude de l'évolution des métriques orientées objet. Deuxièmement, une étude sur les corrélations entre les différentes métriques au niveau des systèmes. Troisièmement, nous mesurons plusieurs attributs de taille, dont le nombre de lignes de codes et le nombre de classes, pour voir comment les différentes métriques se comportent vis-à-vis de la croissance du système dans le temps.

- Ensuite, une étude dans laquelle nous abordons la qualité selon une perspective externe. Le nombre de fautes déclarées est dans ce cas notre indicateur de qualité de référence. Nous étudions la corrélation entre la fréquence des fautes déclarées et les différentes métriques, pour voir dans quelle mesure nous pouvons arriver à suivre l'évolution de la qualité sous cette perspective.

- Enfin, nous étudions comment les différentes métriques peuvent permettre d'étudier l'application des lois de Lehman. Notre étude couvre quelques-unes des observations de M.M. Lehman.

Une conclusion résume les lignes directrices de ce mémoire et présente certaines questions qui restent ouvertes et qui pourraient faire l'objet de travaux futurs.

SECTION 1 : ÉTAT DE L'ART

CHAPITRE 2 : L'évolution logicielle

CHAPITRE 3 : La qualité logicielle au travers des métriques

CHAPITRE 2
L'ÉVOLUTION LOGICIELLE

L'évolution, au sens large, est un concept inévitable de notre réalité. Tout être vivant évolue, se transforme et s'adapte à la réalité qui l'entoure. Cette évolution est inéluctable et imprévisible en raison des nombreux impondérables qui peuvent influencer son déroulement.

Le concept d'évolution s'applique parfaitement en informatique. Utilisons le cas précis d'un système logiciel activement maintenu et mis à jour. Ce système subit alors des changements de façon continue. On peut dire de ce système logiciel qu'il évolue. En effet, une définition de l'évolution logicielle présentée par C.F. Kemerer (Kemerer, 1999) résume bien ce qu'est l'évolution logicielle : « The dynamic behaviour of software systems as they are maintained and enhanced over their lifetimes ». L'évolution logicielle est ainsi liée au comportement dynamique d'un système logiciel au cours du temps, et c'est pourquoi nous pouvons dire du système logiciel changeant de façon continue qu'il évolue.

Il n'est toutefois pas automatique qu'un système logiciel sera l'objet d'une évolution. Ce n'est qu'un type particulier de système logiciel qui sera touché par l'évolution logicielle. M.M. Lehman est un des chercheurs reconnus par ses travaux dans le domaine de l'évolution logicielle. Au travers de ses travaux, trois types de systèmes logiciels distincts et exclusifs sont considérés : Les types E, S et P. Seul le type E peut être considéré comme sujet à une évolution. Le type S regroupe les systèmes possédant une spécification formelle et qui sont par conséquent mathématiquement représentables, tandis que le type P regroupe les systèmes logiciels pour lesquels une spécification n'est pas possible. Afin d'alléger le contenu de ce mémoire, nous n'allons y considérer que des systèmes logiciels de type E (sujets à une évolution). Ce type de logiciel ne peut échapper à l'arrivée de changements. Plusieurs raisons peuvent l'expliquer. Par exemple, le besoin de satisfaire de nouvelles exigences du

client. Ou encore, la correction d'erreurs apparues tardivement dans le processus de conception du logiciel ou lors de son utilisation.

Le phénomène d'évolution logicielle a d'abord été identifié vers la fin des années 60 sous l'appellation de « software growth dynamics » (dynamique de croissance) (Lehman, 2003). Le terme d' « évolution logicielle » a graduellement pris place avec l'établissement des lois de Lehman dans les années soixante-dix (Lehman, 2003). Le fruit des recherches de M.M. Lehman est souvent cité comme référence dans les travaux récents du domaine (Zhang, 2010; Godfrey, 2008; Ali, 2009; Lee, 2007; Mens, 2008a; Xie, 2009).

Avec l'entrée en force du développement de logiciels *open-source* et les moyens techniques disponibles pour effectuer une gestion des versions d'un système logiciel au cours du temps (sauvegardes temporelles), les recherches liées à l'évolution logicielle ont progressivement gagné en importance au cours des dernières années.

Le présent chapitre est divisé comme suit. En premier lieu, nous présentons comment l'analyse d'un système logiciel peut être faite dans un contexte d'évolution. En deuxième lieu, nous traitons des premiers travaux d'importance dans le domaine et qui sont, encore aujourd'hui, reconnus comme des références en la matière : les lois de Lehman. À la suite de leur présentation, de nombreux travaux reliés à ces lois sont exposés.

2.1 Analyse d'un système dans un contexte d'évolution

Lorsqu'un logiciel est maintenu ou mis à jour de façon constante, on considère qu'il évolue. L'évolution logicielle est l'étude des changements survenant dans un système au cours du temps. Ces changements surviennent de façon continue. Toutefois, suivre l'évolution de tels changements dans un contexte continu peut s'avérer très difficile à effectuer. C'est pourquoi il est important d'utiliser des techniques éprouvées pour y arriver. La présente section introduit différentes techniques pour procéder à une mesure des changements dans un logiciel.

Une modification à un système logiciel dans un objectif de maintenance peut être de multiples natures : on peut parler de maintenance de type adaptatif, perfectif ou correctif (Sommerville, 2007).

- Une maintenance adaptative est requise lorsque l'environnement autour du système logiciel est modifié. Ce peut être, par exemple, un nouveau système d'exploitation, ou encore un changement dans la machine qui supporte le système. Ce type de maintenance correspond à environ 18% des coûts de maintenance totaux d'un système logiciel.
- Une maintenance perfective est effectuée pour répondre à des changements dans l'organisation (ou l'entreprise) utilisant ce système. Le système doit donc être adapté pour répondre aux nouvelles exigences. Les changements requis pour une telle maintenance sont souvent de taille supérieure à ceux des autres types de maintenance. Ce type de maintenance correspond à environ 65% des coûts de maintenance totaux d'un système logiciel.
- Une maintenance corrective est requise pour corriger des erreurs dans le code source. Une faute peut habituellement être corrigée très rapidement et sans impact majeur. Toutefois, lorsque les erreurs sont directement reliées aux exigences initiales, les coûts peuvent alors grimper en flèche. Ce type de maintenance correspond à environ 17% des coûts de maintenance totaux d'un système logiciel.

Certains travaux traitent d'un quatrième type : la maintenance préventive (Godfrey, 2008). Celle-ci est utilisée pour trouver les problèmes éventuels avant qu'ils ne deviennent de réels problèmes. Ce type de maintenance n'est pas en soit essentiel au bon fonctionnement du système logiciel. Toutefois, étant donné que dans certains cas jusqu'à 90% du coût d'un logiciel peut être en lien avec l'évolution (Erlikh, 2000), diminuer les coûts de maintenance éventuels peut s'avérer d'une grande importance.

Ainsi, un système logiciel peut être l'objet d'une maintenance pour de nombreuses raisons. Toutefois, analyser un système logiciel au travers de son évolution est une

tâche complexe. Plusieurs travaux présentent des techniques pour analyser un système logiciel dans un tel contexte. Le reste de la présente section traite de ces techniques.

L'étude d'attributs dans le contexte d'un système évolutif se fait d'une façon semblable d'une recherche à l'autre. L'évolution du système logiciel étant continuelle, et parce qu'il serait inconcevable d'étudier le système logiciel à chaque instant sur le plan continu, des captures régulières sont effectuées pour représenter le plus fidèlement possible le système en évolution. Des mesures sont ensuite prises sur chacune de ces captures et les valeurs obtenues sont analysées au travers de courbes (évolution des métriques) ou de corrélations.

W. Ambu et al. (Ambu, 2006) étudient le système JAPS (Java Agile Portal System) sur une période de près d'un an, au travers de 20 itérations. La valeur moyenne de certaines métriques orientées objet (DIT, LCOM, WMC, RFC, CBO et NOC)[1] est prise pour l'ensemble des classes de chaque capture, tout au long des itérations. Les tendances suivies par les courbes sont interprétées.

S. Ali et O. Maqbool (Ali, 2009) mesurent plusieurs indicateurs d'évolution (nombre d'ajouts, de suppressions et de modifications). Leur étude est faite sur plusieurs systèmes logiciels. Les résultats obtenus leur permettent de conclure en la pertinence d'utiliser ces indicateurs au travers de versions successives pour avoir un portrait global de l'évolution.

H. Zhang et S. Kim (Zhang, 2010) comptabilisent le nombre de fautes déclarées pour plusieurs composants d'Eclipse et de Gnome par intervalle de temps afin d'en détecter des patterns généraux dans les courbes formées avec ces données. Six patterns sont observés dans les fréquences de fautes : la décroissance, la croissance, l'impulsion, la colline, les petites variations et les montagnes russes. Ces patterns illustrent dans quelle mesure la qualité est sous contrôle.

---

[1] Ces métriques seront définies plus loin dans le mémoire.

L. Yu, S. Ramaswamy et A. Nair (Yu, 2011) utilisent eux aussi le nombre de fautes déclarées, mais cette fois pour les *releases* (versions) officiels de plusieurs systèmes logiciels (6). Les corrélations sont étudiées entre les résultats obtenus pour les fautes de chaque version et certains attributs : les variations de taille et de complexité. Certaines corrélations en ressortent significatives, mais ce n'est pas le cas majoritairement.

T. Mens, J. Fernandez-Ramil et S. Degrandsart (Mens, 2008a) étudient le comportement dynamique du système Eclipse au travers de ses *releases* officiels majeurs. Des attributs comme la taille, le nombre de changements et la complexité sont mesurés pour chacun des sept *releases* officiels sujets à l'étude. Les tendances de ces valeurs sont analysées pour y observer certaines lois de Lehman (que nous aborderons dans la prochaine section).

A. Murgia et al. (Murgia, 2009) analysent l'évolution de la qualité de deux projets à code ouvert : Eclipse et Netbeans. Les valeurs moyennes de métriques reconnues sont calculées pour chaque classe de chaque système (LOC, RFC, CBO, LCOM) et les tendances des courbes formées par les valeurs prises pour chaque *release* sont analysées. Ces tendances sont ensuite mises en contexte avec le nombre de fautes déclarées pour les mêmes versions.

L'évolution logicielle est très complexe à analyser. Son étude s'avère toutefois pertinente pour mieux comprendre comment certains attributs se comportent dans un contexte évolutif. Pour résumer, l'étude d'un système en évolution se fait en analysant plusieurs captures de celui-ci dans le temps. Les données obtenues peuvent ensuite être analysées selon les tendances qui sont suivies ou encore en étudiant la corrélation avec d'autres attributs mesurés. La prochaine section traite des lois de Lehman, qui sont des travaux majeurs dans le domaine de l'évolution logicielle.

2.2 L'évolution logicielle selon Lehman

Les travaux sur l'évolution logicielle sont relativement récents. Parmi les premiers travaux d'importance qui traitent de l'évolution logicielle se trouvent ceux de M.M. Lehman (Lehman 2003). Ce chercheur a proposé diverses « lois », qui sont en fait des observations sur certaines dynamiques non évitables dans un système propriétaire en constant changement. La présente section traite en détail de ces lois de Lehman. Ces lois sont d'une grande importance dans le domaine du génie logiciel. Elles sont les balbutiements de nombreuses recherches qui sont apparues dans les années qui ont suivi (Godfrey, 2008; Ali, 2009; Lee, 2007; Mens, 2008a; Xie, 2009).

Nous débutons en détaillant les lois de Lehman, c'est-à-dire en les énonçant et en présentant le contexte dans lequel elles ont originairement été présentées. Par la suite, nous traitons des études empiriques reliées aux observations de M.M. Lehman. La présentation de ces études est faite en deux parties. D'abord, par une énumération de certains travaux sur l'application de ces lois, originairement pour les systèmes propriétaires, dans un contexte de systèmes *open-source* (dont le code n'est accessible qu'à la société l'ayant développé). Nous présentons les résultats de ces recherches, en plus de présenter comment les lois de Lehman ont été interprétées au travers des mêmes travaux.

2.3 Présentation des lois de Lehman

Un système logiciel en évolution ne peut échapper à certaines dynamiques. C'est ce qu'a conclu M.M. Lehman au travers de huit observations reliées à l'étude des modifications dans un système. Ces observations sont nommées les Lois de Lehman (Al Ajlan, 2009). Ces lois concernent spécifiquement les systèmes de type E (sujets à une évolution), tels que mentionné précédemment.

À l'origine, les « lois de Lehman » étaient au nombre de trois. Cette quantité a augmenté par la suite, en raison de travaux de recherches et de publications succinctes par M.M. Lehman (Mens, 2008b). Nous allons présenter ces lois sous

deux formes : leur formulation originale (Sommerville, 2007) et l'interprétation simplifiée que nous pouvons en faire.

Première loi, le changement continu : *"A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment"*. Un système logiciel a pour but de répondre à certains besoins. Toutefois, ces besoins vont nécessairement changer. Par conséquent, pour ne pas s'éloigner de ces nouveaux besoins, le système devra être adapté continuellement.

Deuxième loi, la complexité croissante : *"As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure"*. Dans un système logiciel, on retrouve des dépendances complexes entre les différentes entités (classes) qui forment celui-ci. Par conséquent, à mesure que la quantité de ces dépendances augmente, on assiste à une dégradation de sa structure. Autrement dit, la complexité d'un système ne peut faire autrement qu'augmenter. Des opérations de maintenance (maintenance préventive) sont alors nécessaires pour réduire ces dépendances, et donc pour simplifier la structure de ce système.

Troisième loi, l'évolution des systèmes de grande taille : *"Program evolution is a self-regulating progress. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release"*. Les gros systèmes ont une dynamique d'évolution qui leur est propre. Cette dynamique fait en sorte que le nombre de modifications possibles est limité. De plus, certains attributs comme la taille, la période entre deux versions ou le nombre d'erreurs demeurent relativement stables pour chacune des versions.

Quatrième loi, la stabilité organisationnelle (rythme de travail invariant) : *"Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development"*. Tout au long de l'évolution d'un système logiciel, si un changement de ressources ou de personnel a

lieu, celui-ci aura des effets imperceptibles sur le rythme de développement du système logiciel.

Cinquième loi, la conservation de familiarité : *"Over the lifetime of a system, the incremental change in each release is approximately constant"*. Dans un contexte de changements incrémentaux, le nombre d'ajouts apportés à chaque nouvelle version est relativement stable. En effet, dans le cas où un nombre plus grand de fonctionnalités est ajouté dans une version, celle-ci introduira inévitablement une quantité importante de fautes. Par conséquent, les versions successives à cette version servent à corriger les erreurs introduites par ces changements majeurs.

Sixième loi, la croissance continue : *"The functionality offered by systems has to continually increase to maintain user satisfaction"*. Des besoins s'ajoutent constamment à la liste de ceux auxquels doit répondre un système. Par conséquent, la taille de ce système doit inévitablement augmenter pour répondre à ces nouveaux besoins.

Septième loi, la qualité déclinante : *"The quality of systems will appear to be declining unless they are adapted to changes in their operational environment"*. Il est nécessaire d'effectuer des opérations de maintenance dans un système logiciel pour réduire sa complexité. En effet, si aucune opération du genre n'est effectuée, le système se dégradera et par conséquent sa qualité sera en déclin.

Huitième loi, le système de feedback : *"Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement"*. Il est nécessaire d'inclure, dans un système logiciel en constant changement, des mécaniques de feedback pour permettre aux personnes responsables de la maintenance de collecter des informations sur le système (métriques). Sans de tels mécanismes, il sera difficile de garder un œil sur les dynamiques d'évolution du système.

Ensemble, ces « lois » permettent de mieux comprendre les dynamiques existantes dans les systèmes en évolution. Étant donné les nombreux impondérables faisant de l'évolution un phénomène très imprévisible, ces « lois » demeurent des observations. Elles ont été observées par Lehman dans les systèmes logiciels propriétaires (Godfrey, 2001; Al Ajlan, 2009; Herraiz, 2009). De nombreux travaux se sont intéressés à l'application de ces lois dans un contexte de code source ouvert. D'autres travaux ont traité de différentes techniques pour parvenir à observer ces lois. Ces deux ensembles de travaux sont présentés dans la section qui suit.

2.4 Études empiriques relatives aux lois de Lehman

Même si elles ont été formulées pour la première fois en 1974, les lois de Lehman sont encore de nos jours une référence importante pour mieux comprendre les dynamiques d'évolution des systèmes logiciels. Toutefois, les réalités d'aujourd'hui dans le développement logiciel diffèrent de celles présentes dans les années soixante-dix. L'émergence des logiciels libres amène la possibilité de mener des études à très large échelle. Il est donc essentiel de savoir dans quelles mesures les lois de Lehman influent sur les réalités actuelles du développement. Les recherches effectuées en ce sens ont généralement démontré qu'effectivement, ces observations sont constatables tout autant pour un logiciel *open-source* que pour un logiciel propriétaire, et ce, malgré les importantes différences entre les modes de développement de ces deux types. De ces différences, on peut noter par exemple le rythme de croissance qui diffère entre les deux types de logiciels : sous-linéaire pour les systèmes propriétaires (en deçà d'une pente linéaire) et sur-linéaire pour ceux *open-source* (au-delà) (Al-Ajlan, 2009). Il est à noter toutefois que certaines études contredisent les observations d'A. Al-Ajlan. C'est le cas pour celles de S. Sharma et al. (Sharma, 2002) et C. Izurieta & J. Bieman (Izurieta, 2006), dont les travaux ont montré une tendance sous-linéaire pour certains systèmes *open-source* (la même que pour les systèmes propriétaires).

Les résultats répertoriés ci-après présentent l'application de certaines des lois de Lehman dans un contexte de système *open-source*. Un condensé des résultats de ces travaux est présenté dans le tableau 1 (présenté plus loin).

S. Ali et O. Maqbool (Ali, 2009) valident l'application de la cinquième loi sur certains systèmes logiciels *open-source*. Cette loi traite de la conservation de la familiarité (stabilité dans les changements incrémentaux). En utilisant certaines mesures d'évolution (ajouts, modifications, suppressions), ils disposent du matériel nécessaire. Toutefois, la cinquième loi peut être interprétée de multiples façons. Une première interprétation possible consiste à se concentrer sur les changements incrémentaux, c'est-à-dire la variation du nombre de changements d'une version à l'autre. Une autre interprétation possible consiste à calculer le nombre de modifications (ajouts + suppressions + modifications) pour chaque version. Ces deux interprétations donnent des résultats différents. Globalement toutefois, la majorité des systèmes qu'ils ont analysés montrent une certaine constance linéairement pour les changements incrémentaux, comme énoncé dans cette cinquième loi de Lehman.

Y. Lee, J. Yang et K.H. Chang (Lee, 2007) abordent plusieurs des lois de Lehman (1, 2, 6 et 7) et cherchent à savoir dans quelle mesure celles-ci peuvent s'appliquer à un système *open-source*. Le système analysé est JFreeChart. L'applicabilité des lois de Lehman n'est pas l'objectif principal de leur étude, mais les résultats qu'ils ont obtenus leur permettent de les aborder.

- La première loi, le changement continuel, est vérifiée en constatant le nombre de classes ajoutées/supprimées d'une version à l'autre.
- La deuxième loi, la complexité croissante, est validée par le fait que la complexité d'un système écrit en Java est fortement dépendante du couplage entre classes. En montrant que ce couplage est croissant, il s'en déduit que la complexité croît conséquemment.
- La sixième loi, la croissance continue, est validée par une analyse du nombre de classes pour chacune des versions.

- La septième loi, la qualité en déclin, n'est toutefois pas concluante pour les systèmes analysés. Le calcul des valeurs de couplage fan-in et fan-out pour les classes ajoutées et celles enlevées permet d'observer la tendance de la qualité. En constatant un couplage fan-in plus élevé pour les classes ajoutées en plus d'un couplage fan-out plus bas, les chercheurs en déduisent que les classes ajoutées sont de meilleure qualité que celles enlevées. Étant donné que ce sont des propriétés favorisant la réutilisation d'une classe, il y a par conséquent une contradiction avec l'énoncé de la 7ème loi.

T. Mens, J. Fernandez-Ramil et S. Degrandsart (Mens, 2008a) abordent les lois de Lehman d'une toute autre façon. Ils analysent le système Eclipse au travers de sept *releases* majeurs, c'est-à-dire de la version 1.0 à 3.3, afin de pouvoir vérifier si certaines lois de Lehman (1, 2 et 6) s'y appliquent. L'étude est menée avec l'aide des courbes formées au travers du temps.

- La première loi, le changement continuel, est abordée par une étude de mesures des changements (additions, suppressions, modifications) pour chaque version officielle. D'autres métriques sont aussi considérées, dont les variations dans le nombre de fichiers et dans la taille des fichiers.
- La deuxième loi, la complexité croissante, est partiellement validée à partir d'indicateurs de qualité (6) qui, ensemble, permettent de considérer cinq types complémentaires de complexité.
- La sixième loi, la taille croissante, est vérifiée par des mesures de taille à différents niveaux de granularité.

J. F. Ramil, A. Lozano, M. Wermelinger et A. Capiluppi (Ramil, 2008) présentent l'applicabilité des lois de Lehman dans le contexte du développement *open-source*. Les huit lois sont abordées. Les conclusions présentées sont une synthèse de résultats présentés par différents chercheurs relativement aux questions qu'amène le développement *open-source*.

- La première loi, le changement continuel, semble bien s'appliquer pour les projets *open-source* matures.
- La deuxième loi, la complexité croissante, n'est pas unanimement constatable dans les travaux étudiés. Dans certains cas, les systèmes présentent une complexité croissante. Dans d'autres cas, une décroissance de la complexité est observée.
- La troisième loi, l'évolution des systèmes de grande taille, ne semble pas être applicable lorsqu'il est question de développement *open-source*. Des facteurs externes peuvent en effet influencer le rythme de croissance et de changement.
- La quatrième loi, la stabilité organisationnelle, est difficilement applicable étant donné que ce sont plusieurs petits groupes de développeurs qui font évoluer un système logiciel avec chacun leur propre politique et leur propre organisation.
- La cinquième loi, la conservation de familiarité, est elle aussi difficilement applicable. Le code source et les documents étant disponibles à tout le monde, y compris les utilisateurs, tous ont une connaissance approfondie du système.
- La sixième loi, la taille croissante, s'applique bien aux systèmes *open-source*.
- La septième loi, la qualité en déclin, n'a pas encore été vraiment testée étant donné que les différents indicateurs possibles sont difficilement mesurables pour les systèmes *open-source*. Cette loi est donc considérée comme possible, mais non validée.
- La huitième loi, le système de feedback, s'applique dans le cas des systèmes *open-source*, mais d'une façon différente de celle des systèmes propriétaires.

G. Xie, J. Chen et I. Neamtiu (Xie, 2009) analysent l'évolution logicielle sous la forme d'une étude à très large échelle : Au total, 653 versions officielles sur une période combinée de 69 ans d'évolution. L'ensemble des lois de Lehman est abordé. Plusieurs lois ont pu être validées parmi celles-ci. Les résultats sont les suivants :

- La première loi, le changement continuel, est validée grâce aux modules ajoutés/enlevés au système.
- La deuxième loi, la complexité croissante, est validée difficilement grâce à des mesures de complexité comme la complexité cyclomatique, le couplage et la taille du système.
- La troisième loi, l'évolution des systèmes de grande taille, est validée en mesurant l'incrémentation du nombre de modules du système.
- La quatrième loi, la stabilité organisationnelle, n'a pas pu être démontrée. Xie et al. ont utilisé comme métriques : le nombre moyen de changements par jour, ainsi que le rythme de changement (nombre de fonctions ajoutées ou changées divisé par le nombre total de fonctions).
- La cinquième loi, la conservation de familiarité, n'a elle aussi pu être démontrée. L'étude de la croissance nette des modules, le rythme de croissance ainsi que le nombre total de changements n'ont pu donner de résultats permettant de valider cette loi.
- La sixième loi, la taille croissante, a été confirmée en étudiant diverses mesures de taille (nombre de lignes de code, de modules, de définitions).
- La septième loi, la qualité en déclin, a été abordée sous deux perspectives : avec des métriques de qualité internes et externes. À l'externe sont utilisés les défauts, obtenus grâce à Bugzilla. À l'interne sont utilisées des métriques de complexité. Toutefois, quelle que soit la perspective, la septième loi n'a pu être confirmée.
- La huitième loi, le système de feedback, n'a aussi pu être confirmée.

Le tableau 1 résume les conclusions présentées dans les travaux qui viennent d'être exposés.

	Ali (2009)	Lee (2007)	Mens (2008)	Ramil (2008)	Xie (2009)
1ère loi Changement	-	Validée	Validée	Validée	Validée
2ème loi Complexité	-	Validée	Non confirmée	?	Validée
3ème loi Grands systèmes	-	-	-	?	Validée
4ème loi Stabilité Organisationnelle	-	-	-	?	Possible
5ème loi Familiarité	Non confirmée	-	-	?	Non confirmée
6ème loi Taille	-	Validée	Validée	Validée	Validée
7ème loi Qualité	-	Non confirmée (à l'opposé)	-	Possible	Non confirmée
8ème loi Feedback	-	-	-	Validée	Non confirmée

Tableau 1 : Résumé des recherches sur l'applicabilité des lois de Lehman aux systèmes open-source.

L'évolution logicielle est une réalité à laquelle les systèmes logiciels ne peuvent échapper. Il est tout d'abord essentiel de savoir comment l'analyser. L'analyse par captures successives permet d'avoir une très bonne idée de l'évolution de certains attributs du système logiciel. Il est aussi essentiel de mieux comprendre les dynamiques qu'elle impose dans un système logiciel. Les lois de Lehman ont été formulées à cet effet. De nombreux travaux récents se sont penchés sur la question. Le prochain chapitre présente un tout autre sujet : la qualité logicielle. Toutefois, l'objectif étant d'étudier l'évolution de la qualité des systèmes orientés objet, nous reviendrons aux notions présentées dans ce chapitre.

CHAPITRE 3
LA QUALITÉ LOGICIELLE AU TRAVERS DES MÉTRIQUES

La demande pour des logiciels de qualité est en forte croissance depuis plusieurs années (Aggarwal, 2007). Il est donc pertinent de chercher à disposer de techniques fiables pour mieux évaluer (comprendre, contrôler et prédire) cette qualité. La qualité logicielle est un concept (complexe et) abstrait. Son interprétation peut aussi être variable, un utilisateur n'ayant pas la même perception de la qualité d'un logiciel qu'un concepteur.

La qualité logicielle peut être exprimée de différentes façons. Notons, par exemple, les travaux de J. Voas et W.W. Agresti (2004) qui abordent la qualité en fonction d'un ensemble d'attributs : la fiabilité, la tolérance aux fautes, la sûreté, la sécurité, la disponibilité, la testabilité et la maintenabilité. Le concept de qualité logicielle couvre donc un large éventail d'aspects, ce qui en fait un concept complexe. Ce chapitre présente en profondeur le concept de la qualité dans le contexte du développement de logiciels orientés objet. Nous débutons par une brève présentation sur ce que sont les métriques et les attributs internes/externes. Nous exposons ensuite différentes façons par lesquelles le concept de qualité logicielle a été présenté dans la littérature et comment il est possible de l'utiliser dans le cadre d'une étude empirique. Enfin, nous regroupons les concepts vus dans ce chapitre et dans celui de l'évolution logicielle pour traiter de l'évolution de la qualité. Ce chapitre a donc comme objectif de faire constater à quel point les métriques sont des outils puissants pour évaluer la qualité (attributs) de systèmes logiciels complexes et d'expliquer comment il est possible de le faire.

3.1 La métrologie pour les systèmes orientés objet

Lorsque vient le temps d'étudier certaines propriétés d'un système logiciel, les concepts de la métrologie s'avèrent essentiels. En effet, comme nous le verrons dans

la suite de ce chapitre, disposer de mesures logicielles dans un contexte d'analyse permet d'étudier des attributs (propriétés) de systèmes logiciels. Ces attributs sont donc en quelque sorte la raison pour évaluer une métrique particulière (Abran, 2010). La taille est un exemple d'attribut logiciel. Cet attribut peut être estimé à partir d'un large éventail de métriques (le nombre de lignes de code dans chaque classe, le nombre de classes, etc.). La relation entre une métrique et un attribut (de qualité en particulier) doit toutefois avoir été validée empiriquement avant d'être utilisée (Sommerville, 2007).

Dans les sous-sections qui suivent, nous présentons dans les détails certains des concepts que l'on vient d'introduire. Nous débutons par définir ce que sont les métriques en partant d'une présentation générale et nous détaillons, par la suite, leur utilisation dans le contexte particulier de l'orienté objet. Enfin, nous présentons les attributs logiciels. Ils sont une notion primordiale pour l'application des métriques dans le contexte de la qualité logicielle.

3.1.1 Définition globale de la métrologie et des métriques

L' « International consensus on the basic and general terms in metrology », définit la métrologie comme « la science de la mesure et de ses applications » (Abran, 2010). Les métriques y sont les mesurandes, c'est-à-dire la grandeur à mesurer. Ensemble, ces deux concepts permettent de travailler avec un concept comme l'évaluation de la qualité.

D'autres définitions des métriques peuvent être formulées. Elles fournissent aussi un élément de mesure pour le logiciel et elles donnent aux attributs des descriptions quantitatives (Honglei, 2009).

Enfin, selon l'IEEE et son « standard of software quality metrics methodology » (Honglei et al, 2009), les métriques logicielles sont une fonction dont l'entrée est le système logiciel et la sortie est une valeur qui peut permettre de décider si un attribut donné affecte le système logiciel en question.

La prochaine section présente le concept de métriques dans un contexte de développement logiciel orienté objet.

3.1.2 Les métriques dans un contexte de développement orienté objet

Nous nous intéressons aux systèmes logiciels développés avec la programmation orientée objet. Plusieurs particularités caractérisent ce type de développement. La principale est l'utilisation des objets, qui sont des entités logicielles représentant chacun un concept distinct du monde réel. Ces objets interagissent entre eux de plusieurs façons: une classe peut faire appel aux services d'une autre classe ou encore elle peut aussi dépendre d'une autre classe par une structure d'héritage. Ces caractéristiques mènent à l'introduction de nouveaux attributs logiciels tels que l'héritage, le couplage entre objets et la cohésion.

Une métrique est donc une mesure quelconque effectuée sur un composant. Une métrique peut bien sûr se baser sur des éléments simples tels que le nombre de lignes de code ou le nombre de méthodes, mais elle peut aussi être complexifiée par des mesures d'éléments beaucoup plus complexes. Les métriques possibles sont par conséquent très variées et nombreuses. La prochaine sous-section présente les attributs logiciels, une notion que nous avons maintes fois mentionnée sans l'avoir encore définie.

3.1.3 Les attributs de logiciels

Les attributs d'un système logiciel sont en quelque sorte des propriétés de ce système. Les attributs logiciels sont divisés en deux catégories : les attributs internes et les attributs externes (Fenton, 1995).

Un attribut interne est purement mesuré en termes du produit même (Fenton, 1995). De tels attributs reflètent donc la structure et l'organisation du code source du programme (Sommerville, 2007).

Un attribut externe est mesuré relativement à comment le produit logiciel interagit dans son environnement avec d'autres entités (Fenton, 1995). Il reflète donc le comportement du système logiciel alors qu'il est en utilisation (Sommerville, 2007).

Une valeur quantitative peut être attribuée aux attributs internes par l'utilisation de métriques. L'association d'une métrique à un attribut interne doit toutefois être validée par de nombreuses études empiriques avant d'être acceptable. V. Basili (Basili,1992) présente un modèle pour y arriver avec le paradigme GQM (Goal-Question-Metrics).

Un attribut externe est évalué à partir d'attributs internes à un système logiciel. Pour l'évaluer, il faut donc déterminer un ensemble d'attributs internes capable d'expliquer l'attribut externe. Ensuite, les métriques associées à ces attributs internes sont calculées et une interprétation est effectuée (Sommerville, 2007). La figure 1 présente des attributs externes et certains attributs internes qui peuvent les représenter.

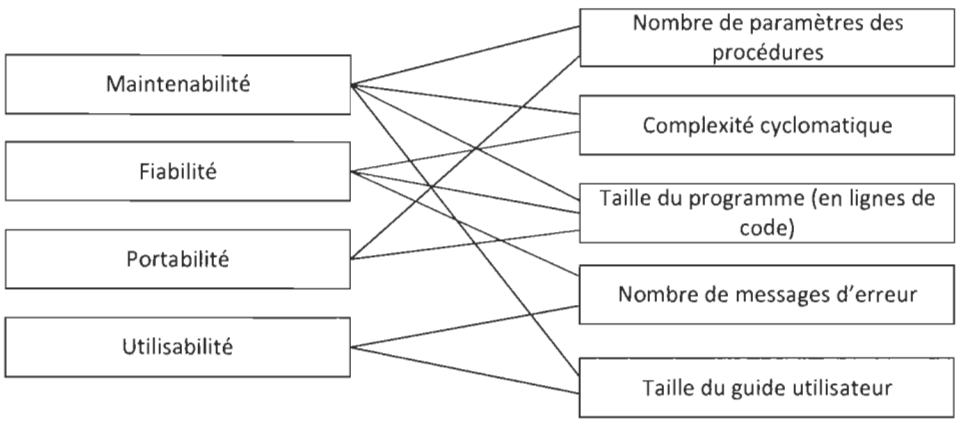


Figure 1 : Liens entre attributs externes et attributs internes (Sommerville, 2007).

Plusieurs métriques distinctes peuvent s'avérer capturer la même information sur un attribut donné (Aggarwal, 2007; Briand, 2000). Une recherche menée par K.K. Aggarwal, Y. Singh, A. Kaur et R. Malhotra (Aggarwal, 2006) illustre bien le fait que de l'information peut être répétée entre plusieurs métriques. Ces chercheurs ont étudié 22 métriques proposées par de nombreux chercheurs afin de procéder à une

large étude empirique (statistiques descriptives, analyse de composantes principales et analyse de corrélation). Ces études menées par K.K. Aggarwal et al. ont permis, grâce à l'analyse de composantes principales, d'extraire un ensemble de 6 métriques qui fournissent l'essentiel de l'information apportée par les 22 métriques regroupées (NOA, NOM, MPC, DAC, LCOM et LCC). De plus, ils ont constaté une forte corrélation entre la taille et presque chacune des métriques, ce qui montre que la taille est un attribut relié à plusieurs autres attributs internes.

La présente section a permis de constater la pertinence de l'utilisation des métriques dans un contexte de développement logiciel orienté objet. Lorsqu'elles sont utilisées d'une façon empiriquement validée, les métriques peuvent être utilisées conjointement pour mesurer certaines dimensions complexes. La prochaine section présente un contexte précis d'application des métriques, c'est-à-dire l'évaluation de la qualité logicielle.

3.2 La qualité logicielle

La section précédente a présenté les métriques logicielles ainsi que leurs relations avec les attributs logiciels. La présente section expose une situation particulière d'application des métriques, c'est-à-dire l'application des métriques dans un concept de qualité logicielle. La qualité logicielle est un concept essentiel en génie logiciel. Il s'agit toutefois d'une notion plutôt complexe, car la définition qui lui est associée varie en fonction du contexte dans lequel elle est formulée. Par exemple, l'utilisateur d'un système logiciel a une différente perception de la qualité comparativement à celle qu'a un concepteur de ce même système. En effet, l'utilisateur est plus sensible à des attributs fonctionnels tels que l'apparition de fautes ou la performance du système logiciel, tandis que le concepteur est davantage sensible à des caractéristiques comme la maintenabilité. La qualité logicielle est présentée dans la littérature au travers d'attributs logiciels de qualité. Cette section présente tout d'abord ces attributs en deux catégories: les attributs internes et les attributs externes. Pour chacune de ces deux catégories, des travaux conséquents sont présentés.

3.2.1 Les attributs de qualité

Parler de qualité pour un bien matériel évoque de multiples interprétations, tout dépendant du type de bien en question. Afin de formaliser la notion de qualité, l'organisation internationale de normalisation (ISO) a établi des normes de qualité ISO (9000-9004) qui définissent entre autres la qualité pour de tels biens matériels.

Par contre, un logiciel est un bien immatériel. Les différences entre ces deux types de biens sont majeures. Un bien matériel existe tout d'abord sous la forme d'un prototype, pour être ensuite fabriqué en série. La qualité peut varier d'un bien à l'autre. Une défaillance dans la machinerie générant ces biens peut conduire à une qualité instable du bien produit. Un bien immatériel, comme c'est le cas pour un système logiciel, n'existe qu'en un exemplaire unique. Si un logiciel est diffusé en plusieurs copies, chacune de celles-ci aura les mêmes défaillances, les mêmes défauts.

Ces différences ont pour effet que les biens matériels et immatériels doivent être considérés différemment. Les définitions auxquelles nous sommes habitués pour les biens matériels ne s'appliquent donc pas aux systèmes logiciels. Une norme ISO existe spécifiquement pour les qualités d'un logiciel. En effet, la norme ISO/IEC 9126 traite de facteurs de qualité tels la fonctionnalité, la fiabilité, la convivialité, l'efficacité, la maintenabilité et la portabilité. Chacune de ces caractéristiques est définie par un ensemble de sous-caractéristiques, calculables avec l'aide de métriques logicielles.

Nous avons introduit dans la section précédente la notion d'attributs logiciels. Certains attributs logiciels permettent de suivre la qualité logicielle. Nous allons nommer ces attributs des attributs de qualité. Comme nous l'avons mentionné précédemment, nous distinguons ces attributs en deux catégories, soient ceux à l'interne et ceux à l'externe. Les prochaines sous-sections décrivent comment ces deux catégories sont abordées dans la littérature. Nous débutons avec les attributs internes de qualité en exposant une suite de métriques reconnue dans la littérature

pour l'estimation de ceux-ci. Nous voyons ensuite les attributs externes de qualité, qui sont estimés à partir des attributs internes.

3.2.1.1 Les attributs internes de qualité et la suite CK

Les attributs internes sont des propriétés de la structure et de l'organisation du code source d'un système logiciel. Certains attributs internes de qualité les plus souvent mentionnés sont le couplage, la cohésion, la complexité, la taille et l'héritage (Dagpinar, 2003; Honglei, 2009; Badri, 2009; Aggarwal, 2006).

Pour évaluer ces attributs internes, une suite de métriques très populaire utilisée à cet effet est la suite de Chidamber et Kemerer (Chidamber, 1994). Cette suite est constituée des six métriques orientées objet que voici :

CBO – Coupling Between Objects (Couplage entre objets) : La métrique CBO calcule pour une classe donnée le nombre de classes auxquelles elle est couplée (et vice-versa).

LCOM – Lack of Cohesion in Methods (Manque de cohésion des méthodes) : La métrique LCOM mesure la dissemblance des méthodes dans une classe. Elle est définie comme suit : LCOM = |P| - |Q|, si |P| > |Q|, où P est le nombre de paires de méthodes qui ne partagent pas d'attributs communs et Q est le nombre de paires de méthodes partageant un attribut commun. Si la différence est négative, LCOM est mise à 0.

DIT – Depth of Inheritance Tree (Profondeur de l'arbre d'héritage) : La métrique DIT mesure pour une classe la longueur du plus long chemin d'héritage en partant de la racine de la hiérarchie d'héritage jusqu'à la classe pour laquelle la mesure est prise (nombre de classes ancêtres).

NOC – Number Of Children (Nombre de classes-enfants) : La métrique NOC mesure simplement pour une classe le nombre de sous-classes immédiates à celle-ci dans la hiérarchie.

WMC – Weighted Methods per Class (Méthodes pondérées par classe) : La métrique WMC donne la somme des complexités des méthodes d'une classe donnée, où chaque méthode est pondérée par sa complexité cyclomatique. Seules les méthodes spécifiées dans une classe sont considérées.

RFC – Response For Class (Réponses à une classe) : La métrique RFC est définie pour une classe par l'ensemble de méthodes qui peut être exécuté en réponse à un message reçu par l'objet d'une classe.

Une autre métrique revient fréquemment lorsqu'il est question d'attributs de qualité. La métrique traditionnelle LOC, qui a été utilisée pour une large quantité d'activités de développement de logiciels différentes, est largement acceptée comme une métrique de taille/complexité (Zhang, 2009). Pour une classe elle compte le nombre de lignes de code. Les métriques de la suite CK ont été validées de nombreuses fois comme de très bons indicateurs de qualité (relativement aux attributs internes de qualité), comme nous allons le voir. Il en est de même pour LOC.

M.-H. Tang, M.-H. Kao et M.-H. Chen (Tang, 1999) ont validé certaines métriques de la suite de Chidamber et Kemerer (CK) comme de bons indicateurs de qualité. La qualité y est représentée par le nombre de fautes, donnée qui est obtenue à partir de rapports de fautes au cours des dernières années. Les fautes ont été classifiées selon trois types : les fautes reliées aux caractéristiques de la programmation objet (héritage, polymorphisme), les fautes reliées à la gestion des objets et les fautes non reliées aux objets.

R. Subramanyam et M.S. Krishnan (Subramanyam, 2003) se sont intéressés à l'utilisation des métriques de Chidamber et Kemerer (CK) et à leur rôle dans la détermination des fautes logicielles. Le nombre de défauts est utilisé comme un

indicateur de la qualité. Certaines métriques de la suite CK se sont avérées pouvoir expliquer les variances dans les défauts. Les résultats se sont aussi avérés différents selon le langage utilisé, c'est-à-dire pour C++ versus Java.

Y. Singh, A. Kaur et R. Malhotra (Singh, 2010) ont investigué sur une possible relation entre les métriques orientées objet de la suite de Chidamber et Kemerer, et la détermination de propension aux fautes à différents niveaux de sévérité des fautes. Une base de données de la NASA est utilisée. Avec les résultats qui ont été obtenus, ils estiment que préciser un modèle selon les niveaux de sévérité peut aider à planifier et exécuter les tests en accentuant les efforts sur les bouts de code qui sont plus propices aux fautes et sur ceux qui pourraient causer les erreurs les plus graves. Les métriques de la suite CK s'avèrent donc efficaces aussi dans ce cas-ci.

De nombreux autres travaux ont été faits sur l'utilisation de la suite de métriques CK, ce qui fait que la littérature sur le sujet est très vaste. R. Subramanyam présente de nombreux autres travaux sur le domaine, dont Briand et al. (Briand, 2000), El Emam et al. (El Emam, 2001) et Chidamber et al. (Chidamber, 1998). Ces nombreux travaux valident aussi, qu'effectivement, les métriques de Chidamber et Kemerer sont de très bons indicateurs de qualité en ce qui a trait à la propension aux fautes et que, par conséquent, elles capturent bien l'information sur les attributs de qualité internes.

La métrique LOC a aussi fait l'objet d'études en ce sens. Par exemple, H. Zhang (Zhang, 2009) a analysé la relation entre la métrique LOC (nombre de lignes de code) et les défauts. En se basant sur deux bases de données de fautes distinctes (celui d'Eclipse et celui de la NASA), il en arrive à des résultats positifs avec l'ensemble des modèles de prévisions utilisés quant à la possibilité de prédire le nombre de fautes à partir de cette métrique.

3.2.1.2 Les attributs externes de qualité

Les attributs externes de qualité correspondent directement aux facettes perçues de la qualité logicielle par les utilisateurs. Il existe de nombreux attributs externes de qualité (Sommerville, 2007) : la compréhensibilité, la robustesse, la fiabilité et l'efficacité sont quelques-uns des attributs externes qui peuvent être formulés. Deux attributs de qualité qui reviennent par contre très fréquemment dans les études empiriques sur le sujet sont la maintenabilité et la résistance aux fautes. Les prochains paragraphes présentent ces deux attributs externes et comment ils sont traités dans la littérature.

La maintenabilité est définie par le « Standard Glossary of Software Engineering » de l'IEEE comme étant la facilité avec laquelle un système logiciel ou un composant peut être modifié pour la correction de fautes, l'amélioration de ses performances (ou d'autres attributs) ou pour l'adapter dans un nouvel environnement (Dagpinar, 2003). Comme nous l'avons vu dans la section précédente, nous traitons les cas où un logiciel doit forcément évoluer. Par conséquent, la maintenabilité s'avère importante à évaluer dans une telle situation.

Le nombre de fautes est souvent utilisé comme un indicateur de bonne qualité (Yu, 2011). Évidemment, plus un système logiciel fera l'objet de fautes, moins il sera considéré comme étant de bonne qualité.

K.K. Aggarwal, Y. Singh, A. Kaur et R. Malhotra (Aggarwal, 2007) ont étudié la relation entre la qualité représentée par différents attributs de qualité (propension aux fautes, maintenabilité, effort et productivité) et la propension aux fautes. L'étude est effectuée sur des systèmes Java (136 classes). L'ensemble des métriques de Chidamber et Kemerer y est utilisé. La régression logistique effectuée a permis de montrer que les métriques sélectionnées arrivent à identifier les classes fautives avec une précision (accuracy) de plus de 80%. Il existe donc une relation entre ces métriques et l'attribut externe représenté par les fautes.

L.C. Briand, J. Wüst, J.W. Daly et D.V. Porter (Briand, 2000) présentent la relation qui existe entre plusieurs mesures d'attributs logiciels (couplage, cohésion, héritage) et la probabilité de détection de fautes pour chacune des classes. Le modèle s'avère au final concluant, avec un taux de classification correcte de 80%.

Y. Zhou et H. Leung (Zhou, 2006) explorent les impacts apportés par la prise en compte de la sévérité des fautes déclarées (sévérité - gravité, ampleur des conséquences - haute/basse) lors de la formation d'un modèle de prédiction des fautes. Les métriques CK sont utilisées comme mesures sur le système. Les différences apportées par une telle distinction se sont toutefois révélées minimes.

Des indicateurs d'attributs de qualité externes autre que les fautes peuvent aussi être utilisés. Un exemple de cela apparaît dans les travaux de M. Dagpinar et J.H. Jahnke (Dagpinar, 2003), dans lesquels ces chercheurs s'intéressent à déterminer quelles métriques orientées objet parmi un ensemble de mesures de taille, d'héritage, de cohésion et de couplage peuvent être des prédicteurs efficaces pour la maintenabilité d'un système logiciel. Dans le cas de cette étude, la maintenabilité est donc l'attribut de qualité étudié et il est évalué par des mesures sur l'historique des rapports de maintenance. Le résultat obtenu est que certaines métriques s'avèrent de meilleurs prédicteurs, comme celles reliées à la taille et au couplage direct de type « import ». D'autres ne révèlent aucun pouvoir prédictif, comme c'est le cas pour l'héritage, la cohésion et le couplage indirect et de type « export ».

Les attributs de qualité sont donc essentiels pour une analyse globale de la qualité d'un système logiciel. L'objectif de ces études est essentiellement d'étudier un attribut externe, et pour ce faire, les attributs internes et un ensemble de métriques sont utilisés. La prochaine section présente une tout autre dimension de l'étude de la qualité logicielle, cette fois dans un contexte de système logiciel en pleine évolution.

3.3 La qualité dans un contexte d'évolution logicielle

Le chapitre précédent a présenté l'étude des dynamiques d'évolution d'un système logiciel. Dans ce chapitre, nous avons vu jusqu'à maintenant comment le sujet de la qualité logicielle est abordé dans la littérature. Nous allons maintenant fusionner ces deux dimensions et présenter l'étude de la qualité logicielle dans un contexte évolutif. Ce sujet a fait l'objet de nombreux travaux, qui sont présentés ci-après.

Certaines métriques reviennent fréquemment pour ce type d'étude. C'est le cas, par exemple, du nombre de fautes (Zhang, 2010; Xie, 2009; Ambu, 2006; Murgia, 2009; Yu, 2011) et de certaines métriques orientées objet (en particulier celles de la suite de Chidamber et Kemerer (Ambu, 2006; Dagpinar, 2003; Murgia, 2009, Jermakovics, 2007; Eski, 2011)).

Étudier comment ces métriques évoluent permet notamment de comprendre et de prédire plus aisément l'évolution de la qualité. Les attributs de qualité ou encore les fautes déclarées sont deux moyens possibles pour arriver à une meilleure compréhension de la qualité. Étudier l'évolution de la qualité d'un système nécessite d'aborder le problème autrement qu'on le fait dans le cas d'une simple version. La littérature présente diverses techniques pour arriver à effectuer une telle étude.

La présente section expose l'essentiel de ces techniques. Premièrement, l'observation de patterns dans l'évolution de certains attributs de qualité permet de mieux comprendre ce phénomène et d'apporter des informations importantes pour l'analyse de la qualité d'un système logiciel. Deuxièmement, en disposant d'indicateurs de référence pour les attributs de qualité, il est possible de valider empiriquement la pertinence de certaines mesures logicielles avec des outils statistiques comme la corrélation. Troisièmement, connaître les variations dans les classes au travers des captures (les classes ajoutées ou enlevées) permet d'analyser autrement l'évolution d'attributs de qualité dans les systèmes orientés objet. Les sous-sections qui suivent exposent certains travaux s'étant intéressés à chacune de ces techniques.

3.3.1 Études de patterns suivis par les attributs de qualité

H. Zhang et S. Kim (Zhang, 2010) étudient l'évolution de la qualité sur une longue période avec l'utilisation des « control-charts » et des patterns de qualité. Ils utilisent le nombre de fautes comme un indicateur de la qualité. Leur étude est effectuée avec deux systèmes logiciels *open-source* : Eclipse (Java) et Gnome (C++). Six patterns communs dans l'évolution de la qualité logicielle sont identifiés :

- Une stabilité de la qualité
- Une qualité croissante
- Un logiciel dont la qualité se détériore avec le temps
- Un pic dans le nombre de fautes, signe d'importantes modifications
- Une colline dans le nombre de fautes
- Une qualité incontrôlable, signe d'une mauvaise qualité.

Ces patterns permettent de comprendre le contrôle qu'ont les développeurs sur un système logiciel. Si un logiciel est de mauvaise qualité, il est très difficile de le maintenir et la fréquence de fautes va fluctuer de façon importante. Par contre, pour un logiciel de bonne qualité, le nombre de fautes sera davantage contrôlé. Selon H. Zhang et S. Kim, l'étude des patterns d'évolution de la qualité est utile pour prioriser les efforts lors de contrôles de la qualité.

Xie et al. (Xie, 2009) mènent une analyse empirique sur l'évolution de sept systèmes *open-source*. Ils enquêtent sur les lois de l'évolution de Lehman, et arrivent à valider certaines de ces lois. Ils constatent aussi que certaines branches des systèmes *open-source* évoluent en parallèle. Ils découvrent des similarités dans les patterns d'évolution des systèmes mis à l'étude. En plus d'utiliser des métriques basées sur le code source, ils recueillent de l'information sur les projets et les fautes afin de procéder à une analyse de la croissance des logiciels, de caractériser les changements de ces logiciels et d'évaluer la qualité logicielle.

W. Ambu et al. (Ambu, 2006) se concentrent sur l'évolution de métriques de qualité dans un projet de type agile/distribué. Le projet a été suivi sur une période régulière, période pendant laquelle ont été collectées des métriques de produit et des métriques de processus. Les métriques de produit incluent la suite de métriques de qualité CK (Chidamber, 1994). En analysant l'évolution de ces métriques, les auteurs évaluent comment la distribution de l'équipe de développement a pu avoir un impact sur la qualité du code source.

Jermakovics et al. (Jermakovics, 2007) proposent une approche permettant d'identifier visuellement des patterns d'évolution de systèmes relativement aux exigences. En fait, ce qu'ils proposent est une visualisation montrant l'évolution d'un système logiciel en parallèle avec l'implémentation des exigences. Ils soutiennent qu'une telle vision peut aider les gestionnaires de projet à garder le processus d'évolution d'un système logiciel sous contrôle. Ils utilisent, dans ce travail, les métriques de complexité, de couplage et de cohésion telles que définies par Chidamber et al. (Chidamber, 1994).

3.3.2 Études statistiques

Pour évaluer les capacités de certaines métriques à suivre l'évolution d'attributs de qualité d'un système logiciel, les études statistiques constituent une piste pertinente. Pour ce faire, l'attribut de qualité est mesuré par une ou plusieurs métriques de référence et la validité des nouvelles métriques est déterminée selon la force de la relation entre ces métriques et la métrique représentant l'attribut de qualité particulier.

Murgia et al. (Murgia, 2009) se concentrent sur l'évolution de la qualité des projets *open-source* conçus selon des pratiques agiles. Ils emploient un ensemble de métriques orientées objet et utilisent la moyenne comme agrégation pour étudier leur relation avec la distribution de fautes au cours d'une période d'évolution. Selon les résultats obtenus, ils concluent qu'aucune métrique prise unitairement ne peut permettre d'expliquer l'évolution des fautes pour les systèmes à l'étude.

Mens et al. (Mens, 2008a) s'intéresse à l'évolution d'Eclipse au travers de mesures prises sur plusieurs releases consécutifs. Sept releases majeurs sont considérés. T. Mens et ses collaborateurs examinent si trois des lois de Lehman (évolution logicielle) sont supportées par les données recueillies. Le focus est mis sur le changement continuel, la complexité grandissante et la croissance continuelle.

Dagpinar et al. (Dagpinar, 2003) examinent la significativité (utilité) de plusieurs métriques orientées objet dans un objectif de prévision de la maintenabilité d'un système logiciel. Les métriques utilisées sont catégorisées selon quatre types d'attributs internes : Les métriques de taille, d'héritage, de cohésion et de couplage. Ils utilisent des données sur l'historique de maintenance de deux systèmes logiciels qui couvrent une période de trois ans.

Lee et al. (Lee, 2007) effectuent un survol de l'évolution de la qualité pour les logiciels libres grâce aux métriques logicielles. Ils soutiennent que les métriques logicielles peuvent être utilisées pour évaluer la qualité tout au long de l'évolution d'un système logiciel. Lee et al. explorent l'évolution d'un système libre en termes de taille, de couplage et de cohésion. Ils discutent aussi des changements survenant pour la qualité logicielle en se basant sur les lois de l'évolution de Lehman (Lehman, 1980; Lehman, 1997). Les métriques logicielles sont calculées sur de nombreux releases des systèmes logiciels libres analysés.

Yu et al. (Yu, 2011) étudient la possibilité d'utiliser les rapports de fautes comme indicateur de qualité. En utilisant certaines méthodes statistiques, ils analysent la corrélation entre le nombre de fautes répertoriées et les changements dans un système logiciel.

3.3.3 Études de modifications (classes ajoutées ou enlevées)

Parce qu'un système logiciel subit des changements de façon continue, certaines classes logicielles deviennent désuètes et sont supprimées en cours de route. D'autres

classes sont aussi ajoutées pour répondre à de nouveaux besoins. Certaines études se penchent spécifiquement sur l'étude des changements qui surviennent relativement aux classes logicielles.

Y. Lee, J. Yang et al. (Lee, 2007) analysent au cours des itérations les classes ajoutées et enlevées, afin de vérifier certaines hypothèses en lien avec l'évolution de la qualité. Des mesures de couplage fan-in, de couplage fan-out et de cohésion sont utilisées pour comparer le groupe de classes enlevées à celui des classes ajoutées au cours du temps.

Eski et al. (Eski, 2011) présentent une étude empirique sur la relation entre les métriques orientées objet et les changements dans un système logiciel. Ils analysent les modifications dans le système tout au long d'une séquence historique de projets *open-source*. Ils proposent une approche basée sur les métriques pour prédire les classes plus sujettes aux changements. Ils utilisent les métriques de la suite QMOOD (Bansiya, 2002) et de la suite CK (Chidamber, 1994).

L'objectif visé par ce chapitre est de présenter la diversité et la complexité du concept de la qualité des logiciels orientés objet. Pour l'analyse d'une simple capture d'un système, la qualité peut être investiguée dans un objectif de prévision ou pour identifier des entités plus à risque d'être problématiques. La qualité peut aussi être analysée au travers de l'évolution d'un système. De nombreuses techniques sont exposées dans la littérature : une recherche de patterns, l'utilisation d'outils statistiques ou encore l'étude des modifications que subit un système au cours du temps. Le prochain chapitre présente les indicateurs de qualité (Qi) (Badri, 2009; Badri, 2011; Badri, 2012). Nous allons exposer leur objectif, comment ils sont calculés ainsi que leurs applications validées dans le domaine du génie logiciel.

SECTION 2 : INDICATEUR D'ASSURANCE QUALITÉ : LE MODÈLE

CHAPITRE 4 : Indicateur d'assurance qualité : Le modèle

CHAPITRE 4
INDICATEUR D'ASSURANCE QUALITÉ : LE MODÈLE

Dans les chapitres qui ont précédé, nous avons présenté deux concepts en lien avec les systèmes logiciels, c'est-à-dire l'évolution logicielle et la qualité logicielle. Nous avons aussi présenté la suite CK, dont les métriques ont été maintes fois démontrées comme étant de bons indicateurs pour plusieurs attributs de qualité. Le présent chapitre introduit le modèle des indicateurs d'assurance qualité (Qi) (Toure, 2007; Badri, 2009; Badri, 2011; Badri, 2012).

Le Qi d'une méthode M_i est définit par une estimation de la probabilité qu'un flot de contrôle parcourra la méthode sans qu'il n'y ait d'échec. Ce modèle peut être considéré comme un indicateur du risque associé à une méthode (et à une classe, à un niveau plus élevé). Le Qi d'une méthode M_i dépend, en fait, de plusieurs caractéristiques intrinsèques de la méthode, comme sa complexité cyclomatique (nombre de chemins de contrôle indépendants), la couverture de son test unitaire (effort de test appliqué sur une méthode), ainsi que le Qi des méthodes invoquées par cette méthode. Nous assumons que la qualité de cette méthode, particulièrement en termes de fiabilité, dépend aussi de la qualité des méthodes avec lesquelles elle collabore (en dépend) pour effectuer sa tâche.

Dans les systèmes orientés objet, les objets collaborent pour assumer leurs responsabilités respectives. Une méthode avec une qualité relativement faible peut avoir (directement ou indirectement) un impact négatif sur les méthodes qui l'utilisent. Il y a ici une sorte de propagation, dépendant de la distribution du flot de contrôle dans un système, qui a besoin d'être capturée. Il n'est pas évident, particulièrement dans le cas de systèmes logiciels orientés objet larges et complexes (flot de contrôle distribué), d'identifier intuitivement ce type d'interférences entre les classes. Ce type d'information n'est pas capturé par les métriques orientées objet traditionnelles.

Le but visé par ce modèle n'est aucunement de capturer l'entièreté de l'information, mais simplement de fournir (de manière intégrée) de l'information habituellement fournie indépendamment par de multiples métriques, qui couvrent chacune une dimension de la qualité (couplage, cohésion, complexité, etc.). L'information capturée est en particulier liée à la facilité de maintenance et à la testabilité.

Nous présentons tout d'abord la construction du modèle, en partant d'une vision globale pour en arriver à une présentation détaillée des principales étapes. Nous traitons par la suite d'un outil (plug-in pour Eclipse) permettant d'évaluer automatiquement les valeurs de Qi pour les méthodes/classes d'un système. Enfin, nous exposons plusieurs recherches menées sur les capacités de ce modèle.

4.1 Construction du modèle

Cette section présente la construction du modèle des indicateurs de qualité. Le but de cette section est de résumer comment, à partir du code source d'un système logiciel, une valeur fixe de Qi est déterminée pour chaque méthode. Cette section reprend certains éléments définis dans le mémoire de F. Toure (Toure, 2007).

Le modèle des indicateurs de qualité est basé sur le concept des graphes de contrôle réduits aux appels (CCG), qui sont une forme réduite des graphes de flot de contrôle traditionnels. Un CCG est, en fait, un graphe de flot de contrôle duquel les nœuds représentant des instructions (ou des blocs d'instructions séquentielles) qui ne contiennent pas d'appels de méthodes sont enlevés.

Les Qi sont normalisés et leur valeur est contenue dans l'intervalle [0,1]. Une valeur faible de Qi pour une classe signifie que cette classe est à risque élevé et nécessite un effort de test (relativement) plus élevé pour assurer sa qualité. Une valeur élevée de Qi pour une classe indique que cette classe est à risque faible (avec une complexité relativement basse et/ou la classe a fait l'objet d'un effort de test relativement élevé – proportionnellement à sa complexité).

La construction du modèle se fait en trois (3) grandes étapes :

- En premier lieu (figure 2 – a), le code source est analysé statiquement et le graphe de contrôle réduit aux appels est généré pour chacune des méthodes.

- En deuxième lieu (figure 2 – b), un système d'équations est généré où chacune des équations correspond à une méthode et est construite grâce au graphe de contrôle réduit aux appels de cette méthode.

- En troisième lieu (figure 2 – c), le système d'équations est résolu et les valeurs des Qi sont obtenues pour l'ensemble des méthodes du système.

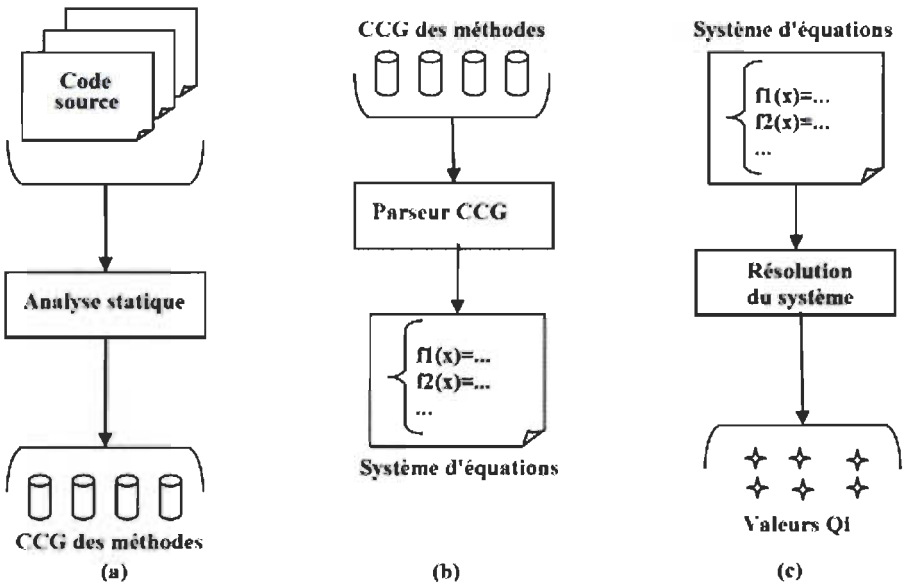


Figure 2 : Passage du code source aux valeurs des Qi.

4.1.1 Analyse du code source pour la création des graphes d'appels (CCG)

La première étape nécessaire à la construction du modèle est l'analyse du code source pour en extraire les graphes de flux de contrôle (CFG), à partir desquels il sera possible de construire les graphes de contrôle réduits aux appels (CCG) pour chacune des méthodes.

Un graphe de flux de contrôle (CFG) est un graphe orienté. Il est formé de nœuds et d'arcs. Ses nœuds représentent soit des structures de contrôle (if-then-else, while, case, etc.), soit une instruction ou un bloc séquentiel d'instructions. Un bloc séquentiel d'instructions est une séquence d'instructions telle que si nous exécutons la première instruction, nous sommes certain que le reste des instructions du bloc sera exécuté et toujours dans le même sens. Un arc orienté lie un nœud N_i à un nœud N_j s'il est possible d'exécuter l'instruction qui correspond à N_j immédiatement après celle associée au nœud N_i. Les arcs du graphe indiquent ainsi le transfert de contrôle d'un nœud à l'autre.

Un graphe de contrôle réduit aux appels (CCG) est un graphe de flux de contrôle (CFG) simplifié par une suppression des nœuds représentant les instructions qui ne conduisent pas à des appels. Un tel graphe (CCG) est donc uniquement formé d'appels de méthodes et de structures de contrôle qui leur sont reliées.

Les trois figures qui suivent (figure 3 a, b, c) présentent la création d'un graphe de contrôle réduit aux appels pour une méthode M. La figure 3(a) présente le code de la méthode M, où les éléments S_i sont des séquences d'instructions ne contenant pas d'appels de méthodes. La figure 3(b) présente la même méthode, mais en ne conservant qu'uniquement les séquences d'instructions contenant des appels de méthode. La figure 3(c) présente le graphe de contrôle réduit aux appels correspondant à la méthode M.

```
M ()
{
   If cond0  Then
      S1;
   Else S2;
   S3;
   M1();

   If cond1 Then
      M2();
   Else M3();

   While cond2
   {
     M4();
     If cond3 Then S4;
     M5();
   }
   M6();
}
```

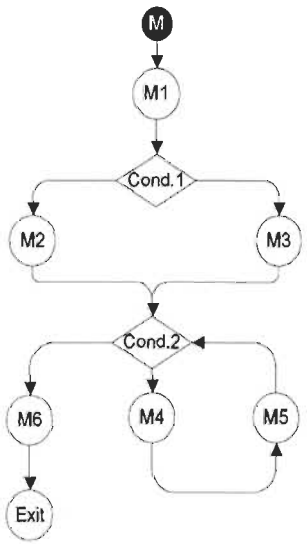
(a)

```
M ()
{
   M1();

   If cond1 Then
      M2();
   Else M3();

   While cond2
   {
     M4();
     M5();
   }
   M6();
}
```

(b)



(c)

Figure 3 : Exemple de construction du CCG d'une méthode (Toure, 2007).

4.1.2 Assignation de probabilités aux chemins

Le graphe d'appels d'une méthode, tel que construit dans la section précédente, peut être vu comme un ensemble de chemins au travers desquels le flot de contrôle peut passer. Le passage dans un chemin en particulier dépend, en fait, de l'état des conditions intégrées à chacune des structures de contrôle. Pour capturer cette caractéristique probabiliste des flots de contrôle, une probabilité est assignée à chaque chemin C d'un graphe de contrôle d'appel de la façon suivante :

$$P(C) = \prod_{A \epsilon \theta} P(A) \qquad (1)$$

θ : L'ensemble d'arcs dirigés composant le chemin C

P(A) : La probabilité d'un arc d'être parcouru à la sortie d'une structure de contrôle existante.

Nœud	Affectation des probabilités
(If, else)	0.5 pour l'arc de sortie condition = vrai 0.5 pour l'arc de sortie condition = faux
while	0.75 pour l'arc de sortie condition = vrai 0.25 pour l'arc de sortie condition = faux
(Do, while)	1 pour l'arc : Les instructions internes sont exécutées au moins une fois
(Switch, case)	1/n pour chacun des arcs pour les n cases
(?, :)	0.5 pour l'arc de sortie condition = vrai 0.5 pour l'arc de sortie condition = faux
for	0.75 pour entrer dans la boucle 0.25 pour sortir de la boucle
(try, catch)	0.75 pour l'arc du bloc try 0.25 pour l'arc du bloc catch
Polymorphisme	1/n pour chacun des n appels éventuels

Tableau 2 : Règles d'affectation des probabilités aux structures de contrôle.

Pour faciliter les expérimentations (simplifier l'analyse et les calculs), nous assignons des probabilités aux différentes structures de contrôle d'un programme Java selon les règles données dans le tableau 2. Ces valeurs sont assignées automatiquement pendant l'analyse statique du code source d'un programme lors de la génération des modèles Qi. De manière alternative, les valeurs des probabilités auraient aussi pu être assignées par des programmeurs (qui connaissent le code) ou être obtenues par une analyse dynamique. L'analyse dynamique est toutefois en dehors de la portée de ce mémoire.

En reprenant l'exemple pratique de la section précédente, quatre chemins peuvent être extraits :

C_1 : M_1, M_2, M_6

C_2 : M_1, M_2, M_4, M_5, M_6

C_3 : M_1, M_3, M_6

C_4 : M_1, M_3, M_4, M_5, M_6

Selon le tableau 2, la probabilité que le flux de contrôle parcourt chacun de ces chemins est évaluée ainsi :

$P(C_1) = 1 \times 0.5 \times 1 \times 0.75 \times 1 = 0.375$

$P(C_2) = 1 \times 0.5 \times 1 \times 0.25 \times 1 \times 1 \times 1 \times 1 = 0.125$

$P(C_3) = 1 \times 0.5 \times 1 \times 0.75 \times 1 = 0.375$

$P(C_4) = 1 \times 0.5 \times 1 \times 0.25 \times 1 \times 1 \times 1 \times 1 = 0.125$

4.1.3 Construction du système d'équations

4.1.3.1 L'indicateur de qualité

Le Qi d'une méthode Mi est donné par :

$$Qi_{M_i} = Qi^*_{M_i} \cdot \sum_{j=1}^{n_i} \left(P\left(C_j^i\right) \cdot \prod_{m \in \sigma_j} Qi_M \right) \qquad (2)$$

Qi_{Mi} : Qi de la méthode M_i

Qi^*_{Mi} : Qi intrinsèque de la méthode M_i

$C_j{}^i$: $j^{ième}$ chemin de la méthode M_i

$P(C_j{}^i)$: Probabilité d'exécution du chemin $C_j{}^i$ de la méthode M_i

Qi_M : Qi des méthodes incluses sur le chemin $C_j{}^i$

n_i : Nombre de chemins du CCG de la méthode M_i

σ_j : Ensemble des méthodes appelées dans le chemin $C_j{}^i$.

En appliquant la formule précédente (2) à chaque méthode du système, nous obtenons un système de N équations (N est le nombre de méthodes du programme). Ce système est non linéaire et est composé de nombreux polynômes à plusieurs variables. Avec l'exemple précédent nous obtiendrons, pour la méthode analysée, l'expression suivante :

$$Qi(M) = Qi_M^* \cdot ([0.375 \cdot Qi(M_1) \cdot Qi(M_2) \cdot Qi(M_6)] + [0.125 \cdot Qi(M_1) \cdot Qi(M_2) \cdot Qi(M_4) \cdot Qi(M_5) \cdot Qi(M_6)] + [0.375 \cdot Qi(M_1) \cdot Qi(M_3) \cdot Qi(M_6)] + [0.125 \cdot Qi(M_1) \cdot Qi(M_3) \cdot Qi(M_4) \cdot Qi(M_5) \cdot Qi(M_6)])$$

4.1.3.2 L'indicateur de qualité intrinsèque

L'indicateur de qualité intrinsèque d'une méthode M_i regroupe plusieurs paramètres caractérisant la qualité prévisionnelle intrinsèque qu'aurait cette méthode si toutes les méthodes qu'elle appelle avaient une fiabilité parfaite. Parmi les éléments qui caractérisent l'indicateur de qualité intrinsèque se trouvent des éléments comme la complexité cyclomatique et l'effort de test unitaire (taux de couverture de test).

L'indicateur de qualité intrinsèque se définit comme suit pour une méthode m_i :

$$Qi_{M_i}^* = (1 - F_i) \qquad (3)$$

Avec $F_i = (1 - tc_i) * cc_i / cc_{max}$, avec

cc_i : Complexité cyclomatique de la méthode M_i

$cc_{max} = max_{1 \le i \le N}(cc_i)$

tc_i : couverture de tests unitaires de la méthode M_i, tci ∈ [0, 1].

Plusieurs études présentent de façon empirique plusieurs évidences sur l'existence d'une relation significative entre la complexité cyclomatique et la propension aux fautes (Aggarwal, 2009; Basil, 1996; Zhou, 2006). Les activités de test réduisent donc les risques inhérents à la complexité d'un programme et par conséquent permettent d'atteindre un bon niveau de qualité. Par ailleurs, la couverture de test fournit une mesure objective sur l'efficacité du processus de test (effort fournit).

4.1.4 Résolution du système d'équations

Le système d'équations tel qu'obtenu précédemment est non linéaire. Pour le résoudre, nous utilisons une méthode itérative nommée la méthode des approximations successives. Avec l'aide de transformations, on peut revenir à un problème du point fixe en définissant la fonction F :

$\mathbb{R}^N \rightarrow \mathbb{R}^N$

$x \rightarrow F(x) = \big(f_1(x), f_2(x), \ldots, f_N(x)\big)$

$avec\ f_1(x) = Qi_i^* \cdot \sum_{j=1}^{n_i} (P(C_j^i) \cdot \prod_{k \in \sigma} x_k)$ (4)

Résoudre le système d'équations en (4) revient à trouver x tel que F(x) = x. L'existence et l'unicité de la solution découle du fait que F est contractante de rapport $K = \max_{i \leq i \leq N} Qi_i^*; K \leq 1$. En posant $x^{(n+1)} = F(x^{(n)})$, on obtient que la suite $(x^{(n)})_{n \geq 0}$ converge vers la solution pour n'importe quelle valeur initiale $x^{(0)}$ de la suite, en particulier pour $x^{(0)}$ = (½, ½, ½, ½, …, ½) : Il s'agit de la méthode des approximations successives. La contraction de F garantit, entre autres, que toutes les composantes de la solution sont donc sur [0,1], c'est-à-dire que les Qi sont des valeurs de l'intervalle [0,1]. La méthode permet d'obtenir rapidement une bonne approximation de la solution du problème.

Nous calculons les N itérations à effectuer pour obtenir une erreur de l'ordre de 10^{-5} grâce à la majoration de l'erreur donnée par : $e_n \leq K^n \left\| 1 - x^{(0)} \right\| \leq K^n \leq 10^{-5}$.
D'où, il suffit de prendre $\geq -5 \cdot \ln(10)/\ln(K)$.

Notons que K est la constante de Lipchitz pour la fonction F définie plus haut. On définira l'indicateur de qualité d'une classe (Qi) comme étant le produit des Qi de ses méthodes. Cette formule est inspirée de l'ingénierie de la fiabilité. Le calcul des Qi ne tient pas compte du flux de données, de ce fait nous mesurerons toujours le rapport entre le flux de contrôle et le flux de données dans un système afin de s'assurer que le flux de contrôle est assez significatif pour pouvoir confronter les résultats de notre modèle avec la réalité lors des expérimentations.

4.2 Présentation du plug-in pour le calcul des Qi

Étant donné la complexité du modèle, une automatisation du processus est nécessaire. Les étapes nécessaires au calcul des indicateurs de qualité des méthodes ont été automatisées grâce à un plug-in mis au point par F. Toure (Toure, 2007). Ce plug-in prend en entrée le code source du système logiciel, et fournit en sortie une feuille Excel contenant l'ensemble des méthodes avec le Qi qui leur est associé. Le mémoire de F. Toure mentionne les nombreux cas d'utilisation traités par le plug-in. Étant donné que cet outil n'a pas été conçu dans le cadre de ce travail, nous allons le considérer comme tel (boîte noire). Quelques points sont, cependant, importants à mentionner sur ce plug-in.

- Pour résoudre le système d'équations, le plug-in fait appel à Scilab, un logiciel libre orienté sur le calcul scientifique.
- Le plug-in se greffe à la plateforme Eclipse, un environnement de développement fort utilisé pour le langage Java.
- Le plug-in permet de définir un taux de couverture pour chaque méthode. Par défaut, celui-ci est fixé à 75%. Étant donné que nous ne faisons pas d'analyse des tests dans nos expérimentations et que nous ne disposons pas de données réelles sur les tests effectués sur les systèmes analysés, la valeur pour l'ensemble des taux de couverture reste fixée (par défaut) à 75%.

4.3 Études empiriques relatives au modèle Qi

Le modèle Qi a été l'objet de quelques études empiriques. Cette section présente quelques unes de ces études.

M. Badri et al. (Badri, 2009) présentent le modèle des indicateurs de qualité sous sa forme actuelle en présentant ses capacités à capturer de façon unifiée plusieurs aspects de la qualité des systèmes orientés objet. En comparant, par une analyse en composants principaux, le modèle à un ensemble de métriques couvrant un large éventail de composantes principales (cohésion, couplage, héritage, complexité et

taille) pour un nombre important de systèmes, ils ont pu démontrer que le modèle de Qi capture une très grande partie de l'information fournie par ces métriques.

M. Badri et al. (Badri, 2011) étudient la relation entre le modèle de Qi et la testabilité des classes. Cette testabilité est déterminée selon les valeurs prises par un ensemble de métriques calculées sur des cas de test JUnit pour plusieurs systèmes. En utilisant une analyse de corrélation, ils parviennent à montrer qu'il existe effectivement une relation significative entre la mesure fournie par les indicateurs de qualité et la testabilité des classes.

M. Badri et al. (Badri, 2012) étudie la relation entre le modèle des Qi et la testabilité des classes, mais cette fois en adressant la testabilité sous la perspective de l'effort de test unitaire. Les classes y sont classifiées selon deux catégories, selon si l'effort de test est considéré comme élevé ou relativement bas. Par une analyse basée sur la régression logistique, ils parviennent à montrer qu'un modèle univarié basé sur le Qi peut permettre de prédire l'effort de test unitaire des classes (un bon taux de classification des classes selon l'effort de test unitaire).

Ainsi, le modèle des indicateurs de qualité Qi, à la base développé entre autres pour supporter le processus de test, s'avère aussi un indicateur significatif de testabilité. Ce modèle sera donc inclus dans nos expérimentations aux côtés des multiples métriques retenues. Le Qi sera alors utilisé dans un tout nouveau contexte d'application, par l'ajout de la dimension relative à l'évolution logicielle. Le prochain chapitre présente les expérimentations menées pour mieux comprendre l'évolution de certains aspects relatifs à la qualité en utilisant les métriques.

SECTION 3 : LES EXPÉRIMENTATIONS

CHAPITRE 5 : Présentation globale des expérimentations

CHAPITRE 6 : La qualité selon une perspective interne

CHAPITRE 7 : La qualité selon une perspective externe

CHAPITRE 8 : Utilisation des Qi pour illustrer les lois de Lehman

CHAPITRE 5
PRÉSENTATION GLOBALE DES EXPÉRIMENTATIONS

Ce chapitre présente les diverses expérimentations menées afin de répondre aux objectifs de notre recherche et particulièrement aux hypothèses formulées initialement. Celles-ci ont principalement pour but de permettre une meilleure compréhension de l'information que peuvent fournir les métriques OO relativement à la compréhension du comportement (évolution) de la qualité d'un système logiciel au cours de son évolution, en particulier sur une longue période de temps. Les expérimentations sont réalisées sur des données relatives à l'évolution de trois systèmes et sont collectées sur une importante période de temps. Ces expérimentations sont essentiellement groupées en trois catégories, chacune exposant une perspective différente de la qualité :

- Les deux premières séries d'expérimentations traitent la qualité selon deux perspectives complémentaires : une perspective interne (structure du code) et une perspective externe (comportement - considérant le système comme une boîte noire). Ces deux perspectives sont respectivement représentées par des attributs internes liés à la qualité et par la fréquence de fautes répertoriées par intervalles de temps.
- La troisième série d'expérimentations traite la qualité selon une perspective dynamique. Elle présente comment les Qi peuvent permettre d'aborder les lois énoncées par M.M. Lehman.

Nos objectifs généraux sont les suivants :

- Évaluer les capacités du modèle des Qi à capturer de l'information relative à l'évolution de la qualité d'un système logiciel selon une perspective interne, c'est-à-dire par l'utilisation de mesures internes des systèmes logiciels reconnues dans la littérature du domaine (attributs internes).

- Évaluer les capacités des métriques logicielles (en particulier du modèle Qi) à capturer de l'information relative à l'évolution de la qualité d'un système logiciel selon une perspective externe. Cette perception de la qualité est évaluée par l'apparition de fautes logicielles dans un intervalle de temps donné.
- Explorer comment les Qi peuvent être utilisés pour l'étude de diverses dynamiques d'évolution, telles que présentées par M.M. Lehman au travers de ses lois, en comparaison avec ce qui peut être observé avec des métriques logicielles plus traditionnelles. Nous allons retenir les lois ayant un lien avec la qualité logicielle et nous allons les analyser en se basant sur des démarches présentées dans la littérature (Ali, 2009; Lee, 2007; Mens, 2008; Ramil, 2008; Xie, 2009).

5.1 Présentation des outils

Pour le calcul des métriques logicielles traditionnelles, nous avons utilisé *Borland Together*[2]. Cet outil intègre l'environnement *Eclipse* et permet la cueillette de valeurs pour un large ensemble de métriques logicielles (les principales métriques liées au paradigme orienté objet).

Eclipse[3] est un environnement de développement (IDE) développé en Java et principalement utilisé pour le développement en Java. Ce projet de la fondation *Eclipse* est *open-source* et ses fonctionnalités peuvent être élargies grâce à l'intégration de plug-ins.

Les valeurs des Qi sont calculées avec le plug-in présenté dans le chapitre précédent. Le taux de couverture de test est fixé à 75% pour l'ensemble des méthodes des systèmes analysés. Nous avons opté pour ce choix car nous n'avons aucune donnée concernant les tests effectués sur les systèmes utilisés au cours du temps (évolution des systèmes).

---

[2] http://www.microfocus.com/products/micro-focus-developer/together/index.aspx/
[3] http://www.eclipse.org/

Les traitements statistiques sont effectués avec l'outil *XLSTAT*[4] développé par *Addinsoft*. Cet outil, intégré à même Excel, permet la génération de rapports détaillés contenant l'analyse statistique demandée ainsi que quelques conclusions qui peuvent en être tirées.

Bugzilla[5] fournit des informations sur le suivi des fautes logicielles. Cet outil est un système de suivi de problèmes développé par l'organisation Mozilla et accessible par une interface web. Il est utilisé par de nombreux organismes réputés, tels *Eclipse*, *Mozilla* et *Red Hat* (*Linux*).

CVS (Concurrent versions system) rend accessible le code source de plusieurs versions/captures d'un même système logiciel. Cet outil est un système de gestion de versions gratuit de type client-serveur. Il permet de conserver l'historique d'un système logiciel par un mécanisme de capture à différents instants de son évolution. Un tel historique permet une consultation aisée des changements, une accessibilité au code à différents instants de sa vie ainsi qu'une gestion aisée des versions. De plus, ce mécanisme rend possible les retours en arrière (*rollbacks*). C'est à partir de ce système client-serveur, utilisé à l'intérieur de l'environnement *Eclipse*, que nous obtenons les différentes versions/captures nécessaires à nos expérimentations.

5.2 Métriques de design orientées objet

Les métriques utilisées dans l'ensemble des expérimentations sont celles de la suite de Chidamber et Kemerer (Chidamber, 1994). Ces métriques ont été maintes fois démontrées comme bons indicateurs de nombreux attributs internes de qualité et sont facilement calculables à partir de plusieurs outils de développement. De plus, ces métriques reçoivent une attention considérable de la part des chercheurs et elles sont de plus en plus adoptées dans un contexte professionnel.

---

[4] http://www.xlstat.com/fr.home/
[5] http://www.bugzilla.org/

La suite de Chidamber et Kemerer (CK) est formée des métriques CBO, LCOM, DIT, NOC, WMC et RFC. À ces métriques sont aussi ajoutées les métriques : LOC (une mesure de taille classique et largement utilisée) et notre modèle des Qi.

5.3 Sélection des systèmes logiciels

Pour qu'un système logiciel soit retenu pour nos expérimentations, il doit répondre aux exigences suivantes :

- Ses archives doivent être disponibles sur une large période de temps et doivent exister en quantité importante. En effet, pour qu'une analyse statistique permette d'obtenir des résultats significatifs, elle requiert un échantillon de taille suffisante.
- Ce système doit avoir été développé en Java, étant donné que notre plug-in est spécifiquement conçu pour analyser des logiciels développés dans ce langage.
- Ce système doit être associé à un répertoire de fautes (du type Bugzilla), et ce répertoire doit contenir un nombre important d'entrées de fautes.

Nous avons sélectionné trois systèmes logiciels qui répondent à l'ensemble de ces critères. Deux de ceux-ci sont des composants d'Eclipse (PDE.UI et JDT.Debug), et le troisième est Apache Tomcat.

Les deux composants d'Eclipse (JDT.Debug et PDE.UI) sont retenus pour nos expérimentations car ils suivent différents patterns de qualité (Zhang, 2010). La qualité de JDT.Debug s'améliore au travers du temps (décroissance dans l'apparition des fautes), tandis que celle de PDE.UI est relativement mauvaise (instabilité dans l'apparition des fautes caractérisée par un pattern de montagnes russes dans les fautes). Les captures de ces deux systèmes ont été obtenues grâce à l'outil CVS. Bugzilla est utilisé par ces deux systèmes pour répertorier les fautes. L'étude de ces deux systèmes est faite en se basant sur des captures mensuelles étant donné

l'importante quantité d'archives disponibles pour ces deux composants. Les releases officiels ne sont donc pas pris en compte pour ces deux systèmes. Cinquante-deux captures sont prises de chacun de ces deux systèmes (pour une période d'évolution couvrant plus de quatre ans).

Le troisième système retenu, Apache Tomcat, est un serveur web « open source » développé par la fondation *Apache Software*. Nous avons basé l'étude de ce système sur les releases officiels de la branche 5.5, introduite en août 2004. La version 5.5.35, la dernière version à ce jour issue de cette branche, a été lancée en novembre 2011. Le code de certains releases n'étant pas disponible, nous disposons de trente-et-une captures de ce système. Le répertoire de fautes utilisé par ce système est *Bugzilla*. Le code source des releases de ce système est disponible sur le site web officiel de la fondation Apache[6].

5.4 Démarches pour la cueillette des données

5.4.1 Obtention du code source des captures/releases

Cette démarche permet d'obtenir l'ensemble des captures/versions associées à un des systèmes sélectionnés. La démarche diffère entre les composants d'Eclipse (JDT.Debug, PDE.UI) et Apache Tomcat. La nécessité de disposer de deux démarches est reliée au fait de permettre l'obtention, avec chacun des systèmes, d'une quantité suffisante de captures/versions pour pouvoir effectuer une étude statistique.

1. Cueillette du code source
    a. *Avec les systèmes pour lesquels la cueillette est basée sur des releases officiels (Apache Tomcat)*
        i. Cueillette du code source associé à chacun des releases via les sites web associés

[6] http://archive.apache.org/dist/tomcat/tomcat-5/

ii. Complétion des dépendances (*.jar) manquantes.

b. Avec les systèmes dont la cueillette est basée sur des captures mensuelles (PDE.UI et JDT.Debug)

i. À partir du système de gestion de version (CVS), capture de l'archive correspondant à la dernière version disponible en date du 1er de chaque mois (par exemple, pour une capture du mois d'avril (1 avril), la capture la plus récente disponible pourrait être celle du 27 mars).

ii. Complétion des dépendances (*.jar) manquantes.

5.4.2 Calcul des métriques de classes pour chaque capture (CK, LOC et Qi)

Cette démarche permet d'appliquer l'ensemble de métriques précédemment sélectionnées aux captures/versions d'un système logiciel. Nous utilisons *Together* pour calculer les valeurs des métriques traditionnelles et le plug-in présenté pour le calcul des Qi. Les données associées à ces deux ensembles de données, enregistrées dans deux feuilles *Excel* distinctes, sont combinées en une seule.

Pour ce faire, pour chacune des captures, nous effectuons les étapes suivantes :

1. Lancement de *Borland Together* (outil pour calculer les métriques)
2. Sélection des métriques logicielles à calculer (Suite CK, LOC)
3. Calcul des métriques sur les classes
4. Exportation des résultats dans un fichier *Excel*.

Le plug-in pour calculer les Qi utilise la liste de classes générées avec *Together*. Il génère une nouvelle feuille *Excel* contenant l'ensemble des classes ainsi que le Qi de chacune de celles-ci. Pour ce faire, pour chacune des captures, nous effectuons les étapes suivantes :

1. Exécution du plug-in
2. Récupération de la feuille *Excel* générée contenant les Qi des classes.

Enfin, la dernière étape est de fusionner les informations contenues dans les deux feuilles (celle de *Together* et celle du plug-in) pour chacune des captures. Deux décisions particulières ont été prises.

- Certaines classes n'ont pas de valeur calculée (nul) pour la métrique LCOM (ceci est essentiellement du à la définition même de la métrique). Étant donné que ces classes correspondent principalement à des interfaces ou des classes hors de l'intérêt de cette étude, nous avons supprimé ces classes.
- Pour certaines classes, aucune valeur n'a pu être calculée pour les Qi. Cela est dû au fait que certaines entités ont été considérées par *Together* comme des classes, mais pas par le plug-in de calcul des Qi qui a une définition plus restrictive de celles-ci. Par conséquent, les classes où aucune valeur de Qi n'a été calculée ont été supprimées.

5.4.3 Calcul des métriques au niveau système

Pour une capture donnée (niveau système), la valeur prise par une métrique est obtenue avec une agrégation par la moyenne des valeurs au niveau des classes.

5.4.4 Calcul des métriques de taille

Les métriques de taille sont calculées au niveau système, sur l'ensemble restreint de classes tel que défini dans la démarche précédente (LCOM et Qi non nuls). Les mesures de taille que nous utilisons sont le nombre de lignes de code (SLOC) et le nombre de classes des captures (SNOC).

1. En disposant de la valeur de SLOC pour chacune des classes, la valeur du système est déterminée par une sommation de ces valeurs.

2. Le nombre de classes (SNOC) est un dénombrement des classes dans la feuille de données.

5.4.5 Calcul de la quantité de fautes liées à une capture

Le nombre de fautes associées aux captures/version d'un système est déterminé par une analyse de son répertoire de fautes.

1- Pour Apache Tomcat, dont les captures sont basées sur les releases officiels

   a. Accès au répertoire de fautes Bugzilla[7]

   b. Accès aux *Graphical Reports* (Information générale sur les fautes répertoriées)

   c. Sélection des bons paramètres

      i. Produit : *Tomcat 5*
      ii. Status : *Closed, Verified, Resolved*
      iii. Resolution : *Fixed*
      iv. Version : Cueillette individuellement pour chaque version
      v. Severity : Tous à l'exception de *Enhancement*
      vi. Horizontal Axis : *Version*

   d. Sélection d'un sous-ensemble des releases et compilation du nombre de fautes associées à chacun des releases.

2- Pour PDE.UI et JDT.Debug, dont les captures sont prises mensuellement

   a. Accès au répertoire de fautes Bugzilla[8]

   b. Accès aux *Graphical Reports*

   c. Sélection des bons paramètres

      i. Classification : *Eclipse*
      ii. Product : JDT / PDE et Component : Debug / UI

[7] https://issues.apache.org/bugzilla/
[8] https://bugs.eclipse.org/bugs/

iii. Status : *Closed, Verified, Resolved*

iv. Resolution : *Fixed*

v. Severity : Tous à l'exception de *Enhancement*

vi. Horizontal Axis : *Version*

d. Pour la détermination du nombre de fautes associées à chacune des captures

i. Prenons par exemple une capture prise du 27 mai 2005 et supposons que l'itération suivante est une capture en date du 28 juin 2005. Dans la section *Custom Search*, les paramètres à indiquer sont ceux indiqués dans la figure 4 de façon à ne conserver que les fautes qui se trouvent dans l'intervalle de la présente capture. Les résultats obtenus sont ensuite compilés.

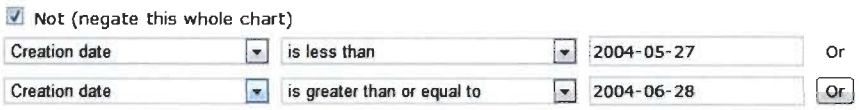


Figure 4 : Paramètres spécifiques pour la recherche personnalisée mensuelle.

5.4.6 Détermination des variations de classes (ajoutées/enlevées) entre les itérations

Nous aurons besoin à plusieurs moments des modifications (ajouts et suppressions de classes) pour chaque itération des systèmes logiciels. Ces données sont obtenues de la façon suivante :

1- Disposition des feuilles Excel associées à chaque capture (contenant les classes, le résultat des métriques de qualité et le calcul des Qi) d'un même système dans un même projet Excel de façon ordonnée.

2- Création d'une macro qui analyse les feuilles de ce projet et qui enregistre les résultats de l'analyse qui suit :

a. Les classes ajoutées pour une itération i sont calculées en déterminant les classes présentes dans l'itération i mais absentes de l'itération i-1

b. Les classes enlevées pour une itération i sont calculées en déterminant les classes présentes dans l'itération i mais absentes de l'itération i+1

5.5 Statistiques descriptives des systèmes analysés

Le tableau 3 qui suit présente certaines informations générales relatives aux captures analysées pour chacun des systèmes. Ces informations sont simplifiées ici pour faciliter la lecture. Elles sont disponibles en intégralité dans l'annexe 1.

Pour chaque capture temporelle d'un système logiciel, nous avons mesuré les valeurs prises par chacune des métriques retenues. Le tableau 4 présente les données relatives aux métriques de classe. Toutefois, comme pour le précédent tableau, l'intégralité des résultats moyens pour chaque capture/version est disponible dans l'annexe 1.

Enfin, pour les mêmes captures, nous avons pris quelques mesures au niveau système. Le tableau 5 présente les données relatives au nombre de lignes de code de chaque capture/version, le nombre de classes ainsi que le nombre de fautes comptabilisées qui leur est assigné. L'ensemble des résultats est disponible dans l'annexe 1.

Il est à noter que le nombre variable de fautes pour PDE.UI (valeur moyenne supérieure à la valeur pour celle de la première et la dernière version) est principalement due à l'apparition d'un pic important de fautes survenant au milieu de la période d'analyse.

Programme	Période couverte	Releases /Captures	Première version analysée			Dernière version analysée		
			Version	Date	Taille (SLOC)	Version	Date	Taille (SLOC)
Eclipse JDT.Debug	4.33 ans	52 (mensuels)	-	2002-03-28	12 201	-	2006-06-26	31 884
Eclipse PDE.UI	4.33 ans	52 (mensuels)	-	2002-08-29	9 519	-	2006-11-28	79 548
Apache Tomcat	7.2 ans	31 (officiels)	5.5.0	2004-08-31	126 927	5.5.35	2011-11-08	170 990

Tableau 3 : Statistiques descriptives relatives à l'évolution de JDT.Debug, PDE.UI et Apache Tomcat.

	Eclipse JDT.Debug			Eclipse PDE.UI			Apache Tomcat		
	Première version	Dernière version	Moyenne	Première version	Dernière version	Moyenne	Première version	Dernière version	Moyenne
Qi	0.791	0.796	0.7945	0.774	0.725	0.716	0.743	0.735	0.742
LOC	96.1	146	138	78.7	119	111	152	154	152
CBO	10.3	14.1	13.2	2.61	23.2	14.6	8.97	9.41	9.14
RFC	80.2	88.9	86.1	15.5	93.4	69.4	60.5	65.9	63.5
LCOM	184	358	323	9.71	48.6	35.5	179	184	174
NOCC	2.70	1.95	2.00	1.00	1.07	1.12	0.686	0.661	0.689
WMC	20.0	28.7	27.2	8.48	22.4	16.5	29.8	30.3	29.9
DIT	2.64	2.83	2.56	0.876	3.02	2.49	1.60	1.64	1.63

Tableau 4 : Statistiques descriptives relatives aux métriques de classes de JDT.Debug, PDE.UI et Apache Tomcat.

	Eclipse JDT.Debug			Eclipse PDE.UI			Apache Tomcat		
	Première version	Dernière version	Moyenne	Première version	Dernière version	Moyenne	Première version	Dernière version	Moyenne
SLOC	12 201	31 884	24 230	9 519	79 548	45 540	127k	171k	153k
SNOC	127	218	173	121	670	384	837	1 108	1 005
Fautes logicielles	114	24	36.8	23	47	50	6	0	16.8

Tableau 5 : Statistiques descriptives relatives aux métriques mesurées au niveau système de JDT.Debug, PDE.UI et Apache Tomcat.

CHAPITRE 6
LA QUALITÉ SELON UNE PERSPECTIVE INTERNE

Nous allons débuter par une investigation sur comment les Qi (et les métriques de design OO) peuvent être utilisées (leur capacité) pour comprendre l'évolution de la qualité logicielle selon un point de vue interne. Nous avons utilisé les métriques OO pour capturer l'évolution de plusieurs attributs internes de logiciels (couplage, cohésion, héritage, complexité et taille). Nous pensons que les Qi sont capables de capturer de l'information à propos de l'évolution des métriques OO. Nous avons appliqué les métriques sélectionnées aux classes de plusieurs captures temporelles des trois systèmes sélectionnés pour notre étude et avons calculé les valeurs moyennes de ces métriques, de façon à se faire une idée sur comment elles évoluent au travers d'une longue période d'évolution pour les trois systèmes.

Pour investiguer si les Qi (et les métriques de design OO) peuvent être utilisés pour comprendre (capturer) l'évolution de la qualité logicielle selon une perspective interne, nous divisons notre analyse en deux études principales. Premièrement, nous présentons (et analysons) l'évolution des Qi en parallèle avec celle des métriques OO sélectionnées. L'objectif est d'observer comment les Qi se comportent relativement à ces métriques (tendance). Nous étudions aussi les corrélations entre les Qi et les métriques OO (au niveau système). Deuxièmement, nous analysons la croissance des classes des trois systèmes à l'étude. La taille est mesurée en utilisant plusieurs métriques variées : Lignes de code du système, Nombre de classes, Lignes de code des classes, Nombre d'opérations par classe (NOO).

6.1 Les hypothèses

En se basant sur les fondements et les attentes mentionnés ci-dessus, nous avons établi les hypothèses suivantes :

Hypothèse 1 : L'évolution des métriques orientées objet sélectionnées au travers de l'ensemble des captures analysées d'un système sera reflétée positivement dans les valeurs des Qi.

Hypothèse 2 : La croissance des classes au travers de l'ensemble des captures analysées d'un système sera reflétée positivement dans les valeurs des métriques OO. Ce sera aussi le cas pour les Qi.

6.2 Évolution des métriques

Nous avons appliqué les métriques sélectionnées aux classes des trois systèmes logiciels à l'étude et nous avons calculé la moyenne des valeurs prises par les métriques pour chaque capture (de chaque système). L'analyse des données collectées nous a permis d'avoir une meilleure idée sur comment les valeurs des métriques OO a progressé pendant la période d'évolution de chaque système. Les figures 5 à 7 présentent les résultats (en termes de l'évolution) pour les trois systèmes à l'étude, avec les Qi et quelques-unes des métriques sélectionnées (présentées avec la normalisation min-max).

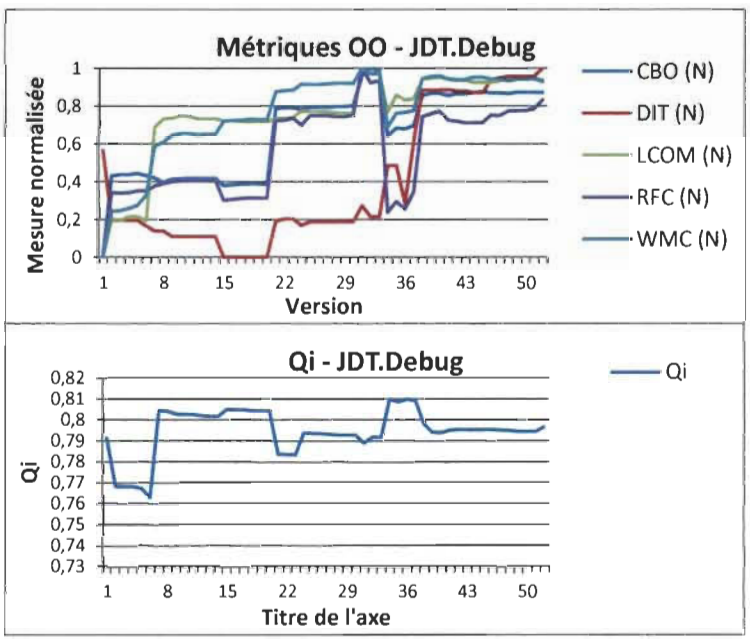


Figure 5 : Évolution des métriques pour JDT.Debug.

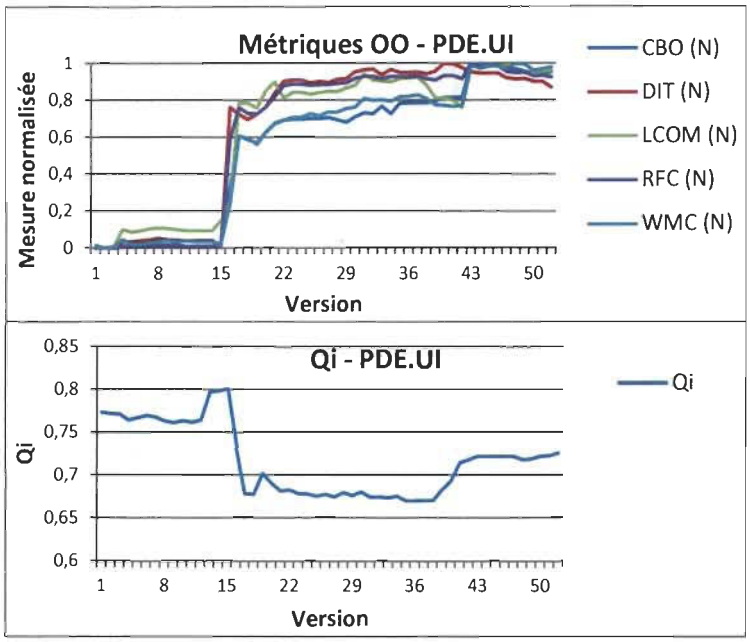


Figure 6 : Évolution des métriques pour PDE.UI.

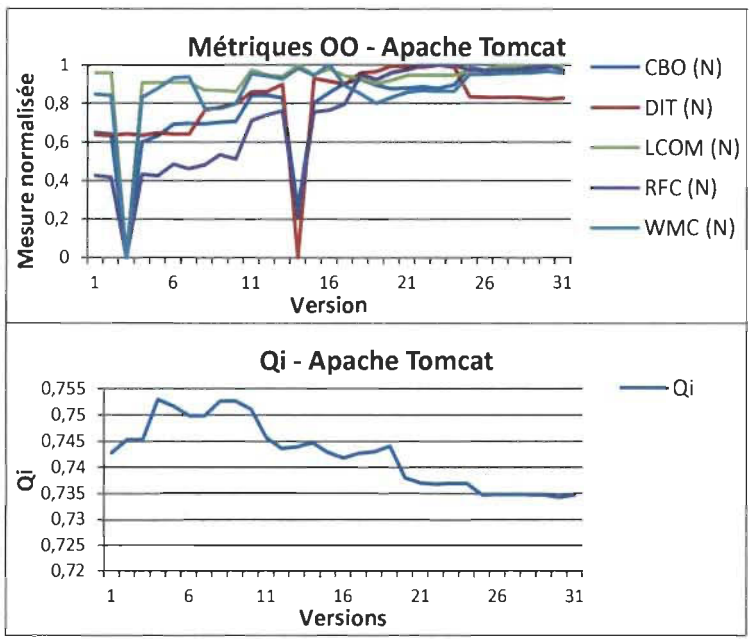


Figure 7 : Évolution des métriques pour Apache Tomcat.

JDT.Debug présente d'importantes variations de croissance et de décroissance. Les deux courbes présentent une évolution sous forme de paliers, c'est-à-dire une période couvrant plusieurs itérations et pendant laquelle les variations demeurent minimes. Les métriques semblent généralement suivre une courbe croissante, tandis que les Qi (malgré une courbe générale qui semble croissante) sont relativement stables avec des hausses importantes neutralisées par des baisses subséquentes quelques itérations plus loin.

PDE.UI présente une uniformité dans les courbes obtenues avec les métriques. Cette tendance est fortement positive, notamment en raison de l'importante hausse survenant à l'itération 16. Cependant, malgré celle-ci, une croissance peut quand même être observée par la suite. Conséquemment, les Qi suivent une tendance contraire. À cette même seizième itération, une baisse importante est constatable pour les Qi, à la suite de laquelle un plateau est alors présent pendant plusieurs itérations.

Apache Tomcat présente aussi une évolution semblable pour l'ensemble des métriques présentées qui se caractérise par une hausse constante et par quelques fluctuations importantes (pics). La courbe des Qi conséquente présente une légère croissance dans les premières itérations, mais qui se transforme en une décroissance aux alentours de l'itération 8. Cette progression négative apparaît de façon relativement continue.

Généralement, nous pouvons observer qu'il n'y a pas de stabilité absolue dans les valeurs obtenues pour les métriques et les Qi. De petites variations apparaissent constamment, ce qui permet de constater une évolution constante (nous reviendrons sur ce point ultérieurement lors de l'étude des lois de Lehman). Les résultats obtenus dans l'ensemble nous portent donc à croire que la métrique des Qi capture (généralement) l'évolution des métriques OO (un indicateur de qualité selon une perspective interne pour un système en évolution). Les prochaines étapes vont pousser l'investigation plus en profondeur avec des analyses de corrélations.

6.3 Corrélations entre la métrique Qi et les métriques OO

L'objectif de cette étape est de déterminer si les Qi reflètent (capturent) l'information obtenue avec les métriques (sélectionnées pour leur capacité à refléter la qualité selon une perspective interne). Nous avons utilisé les corrélations entre les Qi et les métriques sélectionnées pour valider l'hypothèse 1. Le coefficient de corrélation de Spearman (r^2) est une valeur dans l'intervalle [-1, 1]. Plus cette valeur est proche de 1.0 (ou -1.0), plus le lien entre les deux variables est considéré comme fort. Une corrélation entre 0.7 et 1.0 (ou -0.7 et -1.0) est acceptée comme une forte corrélation (Dagpinar, 2003). Les tableaux 6 à 8 présentent les corrélations obtenues. Une corrélation est considérée comme significative lorsque la probabilité que la distribution soit due au hasard ne dépasse pas un certain seuil (fixé dans notre cas à 5%). Ces valeurs sont mises en gras dans les résultats.

JDT.Debug

	CBO	DIT	RFC	NOC	LCOM	WMC	LOC
Qi	**-0.373**	*-0.233*	**-0.443**	*0.062*	*-0.007*	*-0.127*	*-0.138*

Tableau 6 : Corrélations entre le Qi et les métriques OO sélectionnées pour JDT.Debug.

PDE.UI

	CBO	DIT	RFC	NOC	LCOM	WMC	LOC
Qi	**-0.475**	**-0.681**	**-0.516**	**-0.285**	**-0.554**	**-0.539**	**-0.891**

Tableau 7 : Corrélations entre le Qi et les métriques OO sélectionnées pour PDE.UI.

Tomcat

	CBO	DIT	RFC	NOC	LCOM	WMC	LOC
Qi	**-0.854**	**-0.464**	**-0.830**	**0.736**	**-0.747**	**-0.566**	**-0.397**

Tableau 8 : Corrélations entre le Qi et les métriques OO sélectionnées pour Apache Tomcat.

À partir des tableaux 6 à 8, nous avons tiré quelques observations.

- Les corrélations sont significatives avec l'ensemble des métriques pour PDE.UI et Apache Tomcat et de signe négatif (à l'exception de NOC pour Apache Tomcat). Étant donné qu'une forte valeur pour ces métriques est généralement synonyme de plus faible qualité, une forte valeur des Qi signifie donc une qualité plus élevée.
- Les corrélations sont beaucoup plus faibles pour JDT.Debug et ne sont significatives qu'avec CBO et RFC.
- Plus les systèmes sont de grande taille (JDT.Debug < PDE.UI < Tomcat) et plus les corrélations sont fortes entre les Qi et les métriques (à l'exception de DIT et LOC).
- RFC et CBO sont les deux seules métriques dont la corrélation avec les Qi est significative pour les trois systèmes à l'étude. Ces deux métriques sont reliées à deux attributs importants : la complexité et le couplage.

La plupart des corrélations entre les deux groupes sont élevées. Ce qui signifie que les Qi capturent de l'information couvrant plusieurs aspects de la qualité (attributs internes) pour un système en évolution. Toutefois, les résultats provenant de JDT.Debug sont moins concluants. En effet, quelques corrélations significatives émergent pour ce système (CBO, RFC), mais la force des relations y sont pour la plupart plus faibles. Cela peut être dû à la taille du système qui est moins importante que celle des deux autres. Cela peut également être due au fait que les valeurs utilisées sont au niveau système (moyenne des valeurs des classes). De telles constatations valident donc notre première hypothèse.

6.4 Accroissement des systèmes

L'objectif de cette étape est de déterminer si la croissance de la taille au travers de l'ensemble des captures analysées d'un système sera reflétée positivement dans les

Qi et les valeurs des métriques OO. Nous avons utilisé plusieurs indicateurs de taille pour y arriver. Au niveau des systèmes : le nombre de lignes de code du système (SLOC) et le nombre de classes (SNOC). Au niveau des classes : le nombre de lignes de code des classes (LOC) et le nombre d'opérations par classe (NOO). Pour déterminer le lien entre ces indicateurs de taille et les métriques logicielles, nous avons utilisé la corrélation de Spearman sous les mêmes conditions que l'étape précédente. Les tableaux 9 à 11 présentent les valeurs prises par les indicateurs de taille pour la première et la dernière capture analysée des trois systèmes à l'étude. Les tableaux 12 à 14 présentent les corrélations obtenues entre les indicateurs de taille pour l'ensemble des captures de ces mêmes captures et les métriques logicielles.

JDT.Debug

	SLOC	SNOC	LOC	NOO
Première version 2002-04	12200	127	96.1	10.2
Dernière version 2006-07	31900	218	146	13.3

Tableau 9 : Métriques de taille pour la première/dernière version de JDT.Debug.

PDE.UI

	SLOC	SNOC	LOC	NOO
Première version 2002-09	9 520	121	78.7	4.17
Dernière version 2006-12	79 500	670	119	8.86

Tableau 10 : Métriques de taille pour la première/dernière version de PDE.UI.

Apache Tomcat

	SLOC	SNOC	LOC	NOO
Première version 5.5.0	126k	837	152	12.7
Dernière version 5.5.35	171k	1108	154	13.0

Tableau 11 : Métriques de taille pour la première/dernière version de Apache Tomcat.

Pour JDT.Debug, il y a une augmentation du nombre de classes de l'échelle de 72% entre la première itération analysée (127) et la dernière (218). Ce pourcentage se situe à 454% pour PDE.UI (121 à 670) et pour Tomcat, à 32% (837 à 1108). Les figures 8 à 10 révèlent la croissance de la taille ayant lieu pour les trois systèmes. La taille est représentée par les quatre indicateurs présentés précédemment.

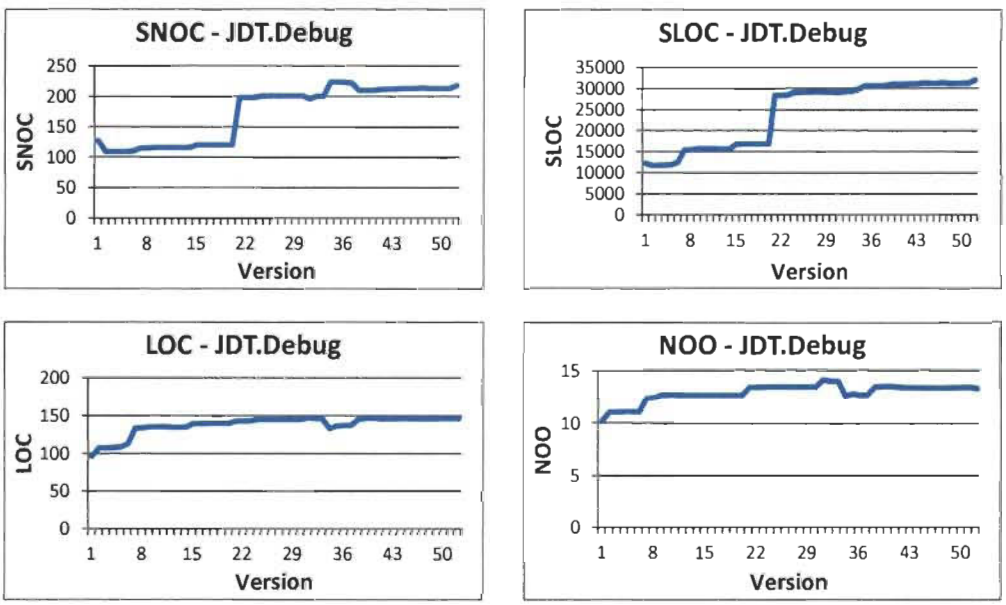


Figure 8 : Évolution de plusieurs métriques de taille pour JDT.Debug.

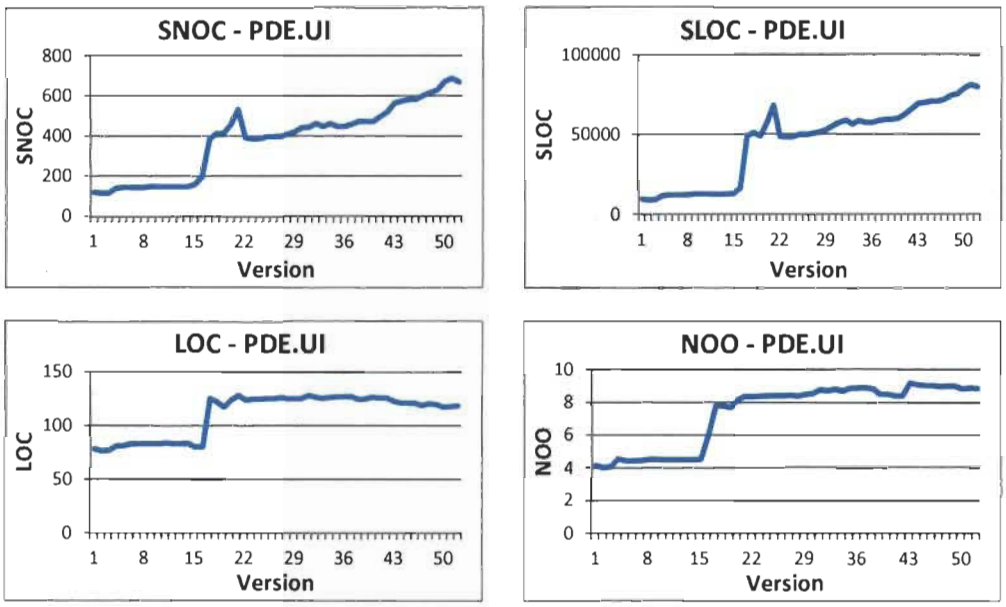


Figure 9 : Évolution de plusieurs métriques de taille pour PDE.UI.

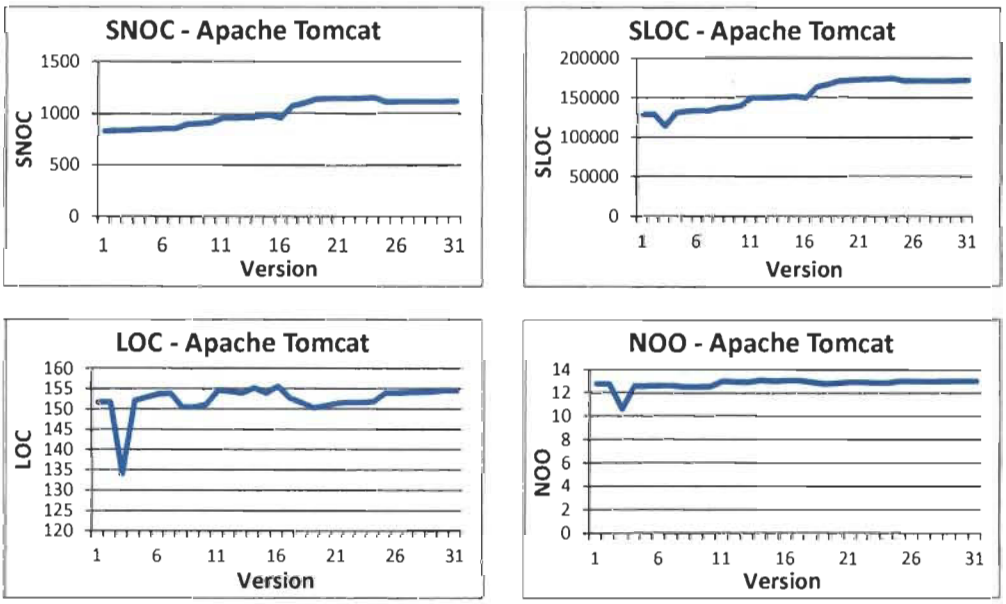


Figure 10 : Évolution de plusieurs métriques de taille pour Apache Tomcat.

Dans les figures 8 à 10, SLOC et SNOC montrent clairement pour l'ensemble des systèmes une évolution croissante de la taille. Cette croissance est relativement linéaire mise à part quelques itérations où l'augmentation est soudaine (ce qui

pourrait être explicable par l'ajout/suppression d'importantes sections de code). Les métriques LOC et NOO ne montrent toutefois pas de résultats aussi clairs, particulièrement pour Apache Tomcat. Le fait que ces deux métriques soient obtenues par une moyenne des données calculées sur les classes peut expliquer cette tendance.

Pour tester l'hypothèse que la croissance des classes au travers des itérations est reflétée par les valeurs des métriques (incluant les Qi), nous avons calculé les corrélations entre les métriques de taille (SLOC, SNOC, LOC et NOO) et la valeur moyenne des métriques sélectionnées relativement aux attributs de qualité (CBO, DIT, RFC, NOC, LCOM, WMC et Qi). Nous pensions qu'il existe une forte corrélation entre ces deux groupes de mesures. Les tableaux 12 à 14 présentent les résultats obtenus. Comme nous le pensions, les corrélations sont généralement très fortes. Les valeurs absolues de ces corrélations sont pour la plupart au-dessus de 0.7, donc elles peuvent être considérées comme fortes. Ces résultats supportent donc fortement la deuxième hypothèse que nous avons formulée. Les corrélations sont moins fortes dans le cas des Qi pour JDT.Debug et PDE.UI, mais elles sont au-dessus de la moyenne pour le plus gros des systèmes analysés (Tomcat). La taille du système semble donc avoir un effet sur les performances du Qi. Mais généralement, les Qi semblent eux aussi très bien capturer l'information relative à la croissance de la taille (surtout au niveau classes – métriques LOC et NOO, ce qui est plus plausible). Ce qui valide la deuxième partie de la deuxième hypothèse.

JDT.Debug

	SLOC	SNOC	LOC	NOO
Qi	*0.129*	*0.219*	*-0.138*	**-0.321**
CBO	**0.795**	**0.654**	**0.845**	**0.732**
DIT	**0.759**	**0.736**	**0.576**	**0.359**
RFC	**0.586**	**0.386**	**0.804**	**0.831**
NOC	*-0.240*	*-0.261*	*0.016*	*0.228*
LCOM	**0.853**	**0.750**	**0.868**	**0.768**
WMC	**0.855**	**0.700**	**0.975**	**0.837**

Tableau 12 : Corrélations entre les métriques de taille et les métriques OO pour JDT.Debug.

PDE.UI

	SLOC	SNOC	LOC	NOO
Qi	**-0.442**	**-0.433**	**-0.891**	**-0.561**
CBO	**0.938**	**0.927**	**0.547**	**0.951**
DIT	**0.750**	**0.735**	**0.797**	**0.801**
RFC	**0.909**	**0.897**	**0.592**	**0.957**
NOC	**0.703**	**0.694**	**0.428**	**0.687**
LCOM	**0.890**	**0.883**	**0.552**	**0.957**
WMC	**0.926**	**0.914**	**0.566**	**0.985**

Tableau 13 : Corrélations entre les métriques de taille et les métriques OO pour PDE.UI.

Apache Tomcat

	SLOC	SNOC	LOC	NOO
Qi	**-0.790**	**-0.736**	**-0.397**	**-0.697**
CBO	**0.801**	**0.775**	*0.306*	**0.620**
DIT	**0.734**	**0.785**	*-0.116*	*0.288*
RFC	**0.927**	**0.901**	*0.168*	**0.500**
NOC	**-0.677**	**-0.655**	*-0.083*	*-0.294*
LCOM	**0.458**	**0.358**	**0.798**	**0.855**
WMC	*0.335*	*0.245*	**0.950**	**0.865**

Tableau 14 : Corrélations entre les métriques de taille et les métriques OO pour Apache Tomcat.

6.4.1 Conclusions sur la perspective interne

Cette première partie des expérimentations a donc permis de constater que les métriques OO que nous avons sélectionnées (ainsi que les Qi) permettent de bien suivre la qualité selon une perspective interne d'un système en évolution. Nous avons observé la relation entre les Qi et de nombreuses métriques qui capturent de l'information sur certains attributs de qualité spécifiques. Nous avons aussi porté attention au lien qui existe entre la croissance de la taille et les différentes métriques OO (incluant les Qi). La prochaine section présente une étude semblable, mais cette fois en considérant la qualité selon une perspective externe.

CHAPITRE 7
LA QUALITÉ SELON UNE PERSPECTIVE EXTERNE

Nous savons maintenant que les Qi sont capables de refléter l'évolution de la qualité d'un logiciel tout au long de son évolution selon une perspective interne au travers de nombreux attributs logiciels. Notre intérêt se pose maintenant sur une perspective externe de la qualité, c'est-à-dire en considérant les systèmes comme des boîtes noires. Nous voulons explorer si les métriques peuvent permettre de capturer la qualité sous cette perspective. Considérer la qualité sous cet angle est essentiel, c'est ainsi que l'utilisateur perçoit la qualité d'un système logiciel. Comme indicateur externe de la qualité, nous utilisons le nombre de fautes associées à chacune des captures.

La qualité d'un système logiciel est inversement liée à la quantité de fautes qui y sont trouvées (Singh, 2009; Subramanyam, 2003; Tang, 1999). Nous allons considérer dans cette section deux groupes de métriques. Premièrement, les métriques traditionnelles (CBO, RFC, LCOM, DIT, WMC, NOC, LOC) et les Qi. Deuxièmement, les changements apportés aux systèmes. Nous pensons qu'aux moments où un système subit un nombre plus important de changements, il est plus susceptible de faire l'objet de problèmes (fautes). Nous allons donc tenter de valider deux hypothèses principales.

Hypothèse 3 : Les métriques OO retenues (ainsi que les Qi) permettent de capturer l'évolution de la qualité telle que capturée par la fréquence d'apparitions des fautes.

Hypothèse 4 : Les changements de taille (ajouts/suppressions de classes) permettent de capturer l'évolution de la qualité telle qu'indiquée par la fréquence d'apparitions des fautes.

7.1 Étude du lien entre les métriques logicielles et les fautes par itération

Nous tentons tout d'abord de valider la première des hypothèses formulées. Pour ce faire, nous utilisons le même ensemble de données que pour la section précédente mais pour étudier cette fois-ci la relation qui existe entre ces données et le nombre de fautes par itération. La corrélation est de type Spearman, selon les critères d'analyse qui ont été précédemment mentionnés. Les tableaux 15 à 17 présentent les résultats obtenus. Les résultats significatifs sont en gras.

JDT.Debug

	Qi	CBO	DIT	RFC	NOC	LCOM	WMC	LOC
Faults	*-0.033*	*-0.266*	*-0.244*	*-0.254*	*0.079*	*-0.271*	**-0.328**	**-0.365**

Tableau 15 : Corrélations entre le nombre de fautes et les métriques OO pour JDT.Debug.

PDE.UI

	Qi	CBO	DIT	RFC	NOC	LCOM	WMC	LOC
Faults	*-0.138*	*0.268*	*0.273*	*0.267*	*0.118*	*0.262*	*0.198*	*0.211*

Tableau 16 : Corrélations entre le nombre de fautes et les métriques OO pour PDE.UI.

Apache Tomcat

	Qi	CBO	DIT	RFC	NOC	LCOM	WMC	LOC
Faults	*0.194*	*-0.034*	**0.509**	*0.124*	*0.147*	**-0.372**	*-0.348*	**-0.404**

Tableau 17 : Corrélations entre le nombre de fautes et les métriques OO pour Apache Tomcat.

Nous pouvons clairement constater que les résultats ne permettent pas de tirer aucune conclusion valide. Certaines corrélations significatives ressortent, mais ce ne sont que des cas isolés. Même les Qi, avec lesquels nous pensions obtenir de bons résultats, ne donnent aucun résultat significatif. Nous ne pouvons donc pas valider l'hypothèse 3. Nous pensons que la faiblesse de ces corrélations est due au fait que

nous avons utilisé l'aggrégation par la moyenne pour calculer les valeurs des différentes métriques au niveau système, ce qui peut mener à une perte d'information.

7.2 Étude du lien entre la quantité de changements et les fautes par itération

Nous tentons maintenant de valider notre deuxième hypothèse pour cette section (hypothèse 4). Nous avons mentionné précédemment la démarche à suivre pour obtenir le nombre de changements par itération (ajouts/suppressions de classes) ainsi que celle pour obtenir le nombre de fautes par itération. Nous avons mesuré le lien qui existe entre ces deux variables pour déterminer si l'étude de la fréquence de changements peut être un bon indicateur pour suivre la qualité d'une perspective externe. Ici encore, nous avons utilisé la corrélation de Spearman pour l'étude. Une corrélation supérieure à 0.7 est considérée comme forte et les valeurs significatives sont en gras. Le tableau 18 présente ces résultats.

Corrélations avec Fautes	Nombre d'ajouts	Nombre de suppressions	Nombre d'ajouts + Suppressions
JDT.Debug	*0.152*	**0.306**	*0.144*
PDE.UI	**0.461**	**0.445**	**0.475**
Apache Tomcat	**0.430**	*0.321*	**0.386**

Tableau 18 : Corrélations entre les changements et le nombre de fautes pour JDT.Debug, PDE.UI et Apache Tomcat.

Des corrélations significatives existent donc entre les deux ensembles de variables au niveau système. Elles ne peuvent toutefois pas être considérées comme fortes (inférieures à 0.7) et ne sont pas présentes pour l'ensemble des systèmes. La présence d'une relation entre ces deux variables est constatable et nous pouvons donc affirmer qu'une itération avec un nombre important d'ajouts ou de suppressions (ou de la somme des deux) aura pour effet une baisse de la qualité (selon une perspective externe). Nous avons ainsi pu valider l'hypothèse 4.

7.3 Conclusions sur la perspective externe

Cette section a présenté la qualité sous une perspective externe, c'est-à-dire relativement à la perspective qu'ont les utilisateurs d'un système (boîte noire). Le nombre de fautes par itération n'a pu être capturé par les métriques logicielles de qualité (CBO, RFC, WMC, DIT, LCOM, NOC, LOC) ainsi que par les Qi. Toutefois, l'utilisation des métriques « nombre d'ajouts de classes », « nombre de suppressions de classes » et « nombre d'ajouts + suppressions de classes » a permis de capturer ces informations.

CHAPITRE 8
UTILISATION DES QI POUR ILLUSTRER LES LOIS DE LEHMAN

Cette section présente une autre dimension de la qualité. Plusieurs dynamiques sont inévitables lors de l'évolution d'un système logiciel et celles-ci ont un effet direct sur l'évolution de sa qualité.

La présente série d'expérimentations s'intéresse à l'évolution d'attributs de la qualité en fonction de ces dynamiques. M.M. Lehman a mené plusieurs observations sur l'évolution des systèmes propriétaires (fermés). Ses observations sont connues sous le nom de « Lois de Lehman ». L'applicabilité de ces observations (dynamiques) aux systèmes *open-source* (ouverts) a été abordée dans de nombreux travaux que nous avons précédemment abordés. Les expérimentations qui suivent s'inscrivent dans cette voie.

Notre objectif est d'observer une partie de ces lois, particulièrement celles qui concernent certains attributs de qualité. Nous pensons que les Qi, par le fait qu'ils unifient plusieurs attributs internes de qualité, permettent d'illustrer ces lois.
Nous allons aborder l'ensemble des lois, mais nous n'allons entrer dans les détails que pour certaines d'entre elles. Celles abordées le seront pour leur lien avec des attributs de qualité. Notre objectif général est de valider l'hypothèse qui suit :

Hypothèse 5 : Les Qi vont parvenir à illustrer plusieurs des lois de Lehman (dynamiques d'évolution).

8.1 Démarche globale

Pour chacune des lois de Lehman, la démarche utilisée est semblable et suit les étapes suivantes :

1- Détermination de la pertinence d'illustrer la loi; elle doit concerner les attributs de qualité et doit pouvoir être étudiée avec les données des précédentes expérimentations. Si cette loi n'est pas jugée pertinente, elle n'est pas traitée.

2- Établir une démarche d'expérimentation pour valider la loi, avec des métriques logicielles et avec les Qi. Pour les métriques logicielles, les démarches sont inspirées de travaux parallèles présentés par divers chercheurs sur d'autres ensembles de données. La pertinence d'utilisation des Qi sera validée par des études de corrélations et/ou de courbes.

3- Valider la présence de la dynamique à l'étude à partir des métriques traditionnelles et comparaison des résultats avec ceux obtenus au travers de divers travaux similaires.

4- Valider la pertinence d'utilisation des indicateurs de qualité (Qi).

8.2 Analyse des lois

Les sous-sections qui suivent présentent chacune des lois prises individuellement. Pour chacune de celles-ci, une démarche et une hypothèse précises sont spécifiées.

8.2.1 Le changement continuel (1ère loi)

Un système en évolution subit de nombreux changements. La première loi stipule que ces changements surviennent de façon continue. Plusieurs des travaux sur lesquels nous basons cette étude (Lee, 2007; Mens, 2008; Ramil, 2008; Xie, 2009) ont abordé l'applicabilité de cette loi, et tous ont pu la valider pour les systèmes *open-source*. Cette loi a un effet direct sur la qualité, car les fluctuations de la qualité sont directement liées aux changements apportés au même système. Nous pensons donc que les données dont nous disposons vont nous permettre d'observer une continuité dans les changements apportés aux systèmes logiciels.

Nous allons étudier cette loi sous deux perspectives : en étudiant les changements de type ajouts/suppressions (observation des courbes formées par le cumulatif des

changements), et en observant la continuité des fluctuations apparaissant dans plusieurs métriques OO (métriques de taille, CBO, RFC).

Pour étudier la continuité du changement avec les Qi, nous allons étudier les variations (deltas) dans les valeurs de cette métrique. Pour une version i, la valeur de la variation est calculée ainsi : (Qi associé à la version i) – (Qi associé à la version i-1). Étant donné que les Qi sont calculés par la résolution d'un système d'équations non linéaire dont les variables sont les méthodes appelées, une modification dans un chemin d'appel récurrent sera considérée comme plus importante qu'une modification dans un chemin rare (cela dépend du flot de contrôle). Donc nous devrions capturer la présence de changements dans les fluctuations des valeurs des Qi.

Hypothèse 5 (1) : Les changements apparaissent de façon continue tout au long de l'évolution des systèmes à l'étude.

Pour tenter de valider cette hypothèse, nous allons suivre la démarche suivante :

1. Observation de la première loi à partir des métriques traditionnelles
    a. Obtention du nombre de changements cumulés pour chaque itération du système (ajouts, suppressions) et représentation sous forme graphique de ces données.
    b. Calcul des variations dans les métriques sélectionnées (taille, CBO, RFC) et représentation de ces variations dans un graphe approprié.
    c. Analyse des résultats et tentative de validation de l'hypothèse.
2. Interprétation avec les Qi
    a. Calcul des variations (deltas) des Qi et représentation sous forme graphique de ces données.
    b. Analyse des résultats et tentative de validation de l'hypothèse.

Débutons par analyser la présence de changements continus par les métriques traditionnelles. Les figures 11 à 13 présentent les courbes formées par le nombre de changements cumulés (ajouts et suppressions) tout au long de l'évolution des trois systèmes étudiés. Par la régression linéaire, nous avons calculé les pentes associées à ces ensembles de données. Ces pentes sont positives (ce qui va de soit étant donné qu'il s'agit d'une somme d'éléments positifs). Notre intérêt est donc sur la force de la pente de ces courbes. Les résultats obtenus permettent de voir que les changements importants ont lieu par périodes. Toutefois, des changements en quantité moins importante ont lieu tout au long de la période analysée. Ces observations nous permettent donc de valider la présence continuelle de changement (première loi de Lehman) dans les trois systèmes logiciels analysés. Nous allons toutefois pousser l'analyse un peu plus loin en s'intéressant aux variations apparaissant dans les valeurs d'autres métriques logicielles.

Il est à noter que dans les trois figures, la courbe des ajouts se situe au-dessus de celle des suppressions. Le nombre d'ajouts est ainsi clairement plus élevé que le nombre de suppressions. Nous allons revenir sur ce fait lorsque nous aborderons la sixième loi (croissance continue).

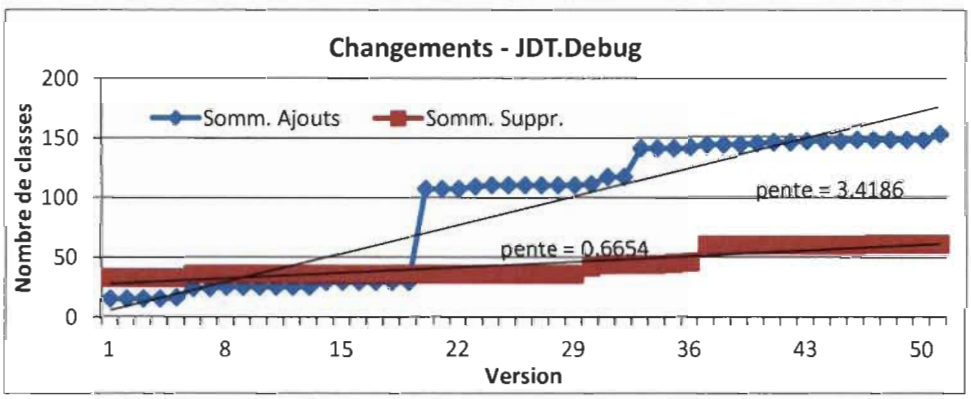


Figure 11 : Évolution du cumulatif de changements pour JDT.Debug.

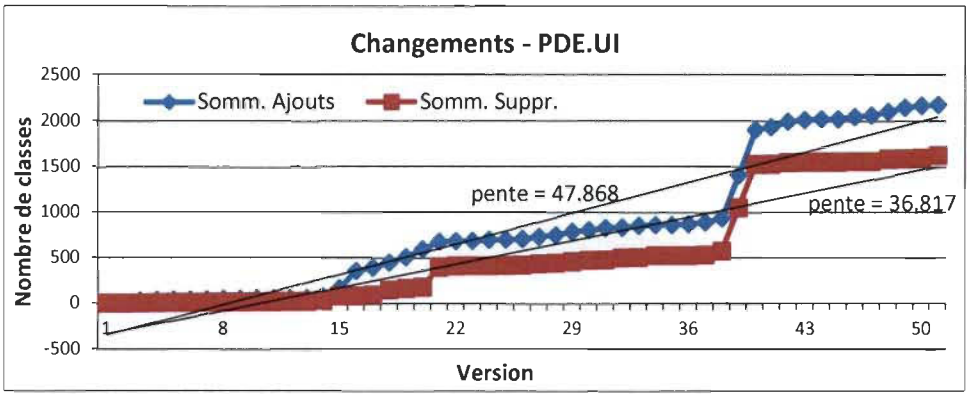


Figure 12 : Évolution du cumulatif de changements pour PDE.UI.

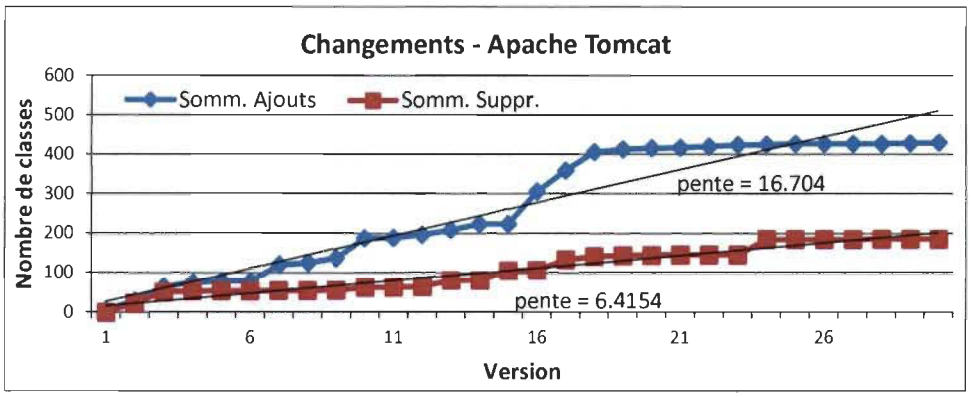


Figure 13 : Évolution du cumulatif de changements pour Apache Tomcat.

L'étude des variations dans différentes métriques est un autre indicateur de changements dans les systèmes à l'étude. En effet, si des variations apparaissent dans ces métriques, c'est qu'il y a eu des changements d'apportés à ceux-ci. Les figures 14 à 16 présentent ces variations pour six métriques logicielles : le nombre de lignes de code du système (SLOC), le nombre de classes (SNOC), le nombre de lignes moyen par classe (LOC), le nombre d'opérations moyen par classe (NOO), le couplage entre objet moyen par classes (CBO) et les réponses pour une classe en moyenne par classe (RFC).

Nous pouvons observer que les variations sont assez dispersées pour l'ensemble des systèmes à l'étude. Par conséquent, il y a effectivement pour ces systèmes logiciels une présence continue de changements tout au long de la période d'étude. Ce qui

confirme une fois de plus que le changement continu est observable avec les systèmes analysés et que par conséquent l'hypothèse 5 (1) est validée.

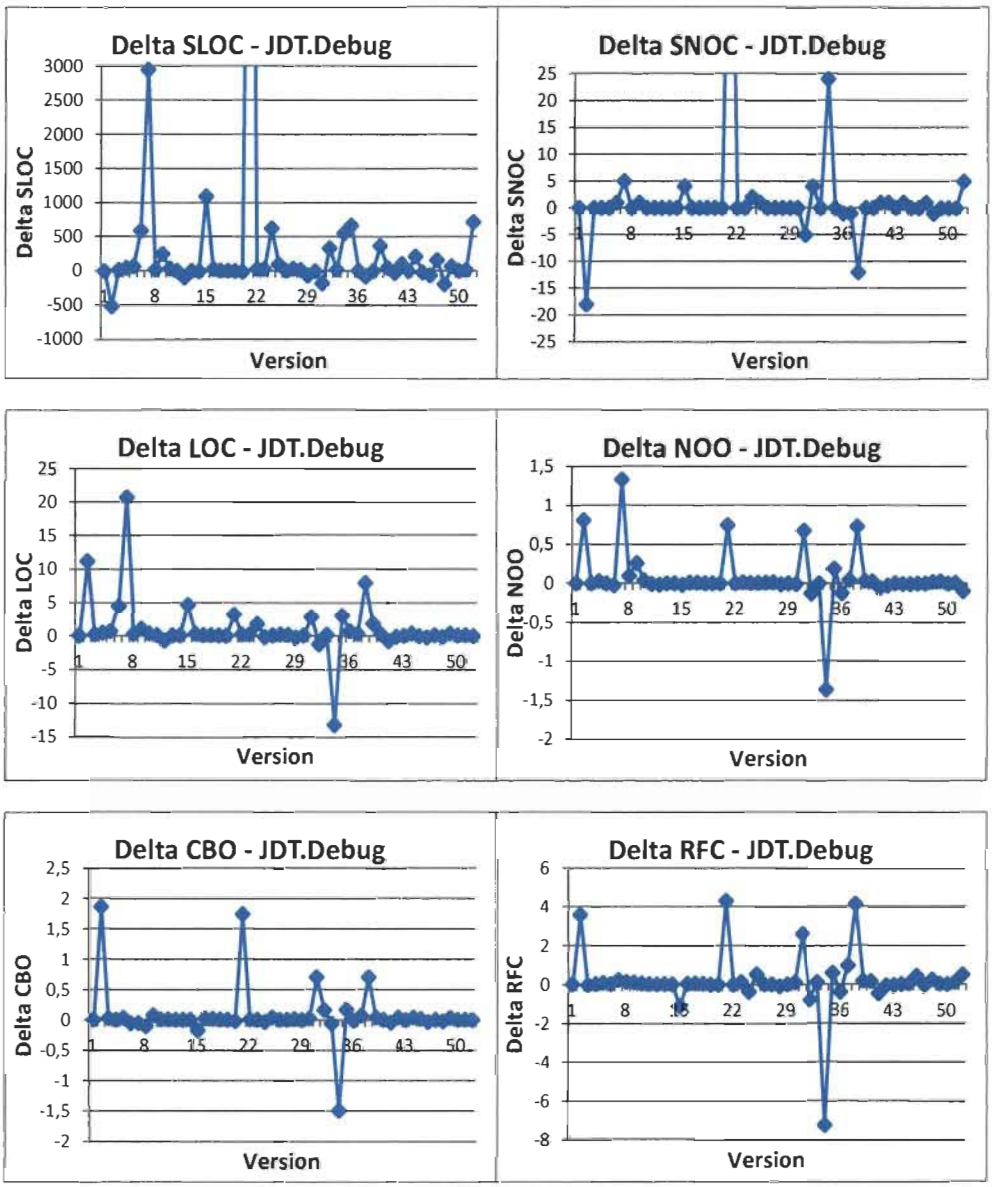


Figure 14 : Évolution des variations subies dans les métriques de taille, CBO et RFC pour JDT.Debug.

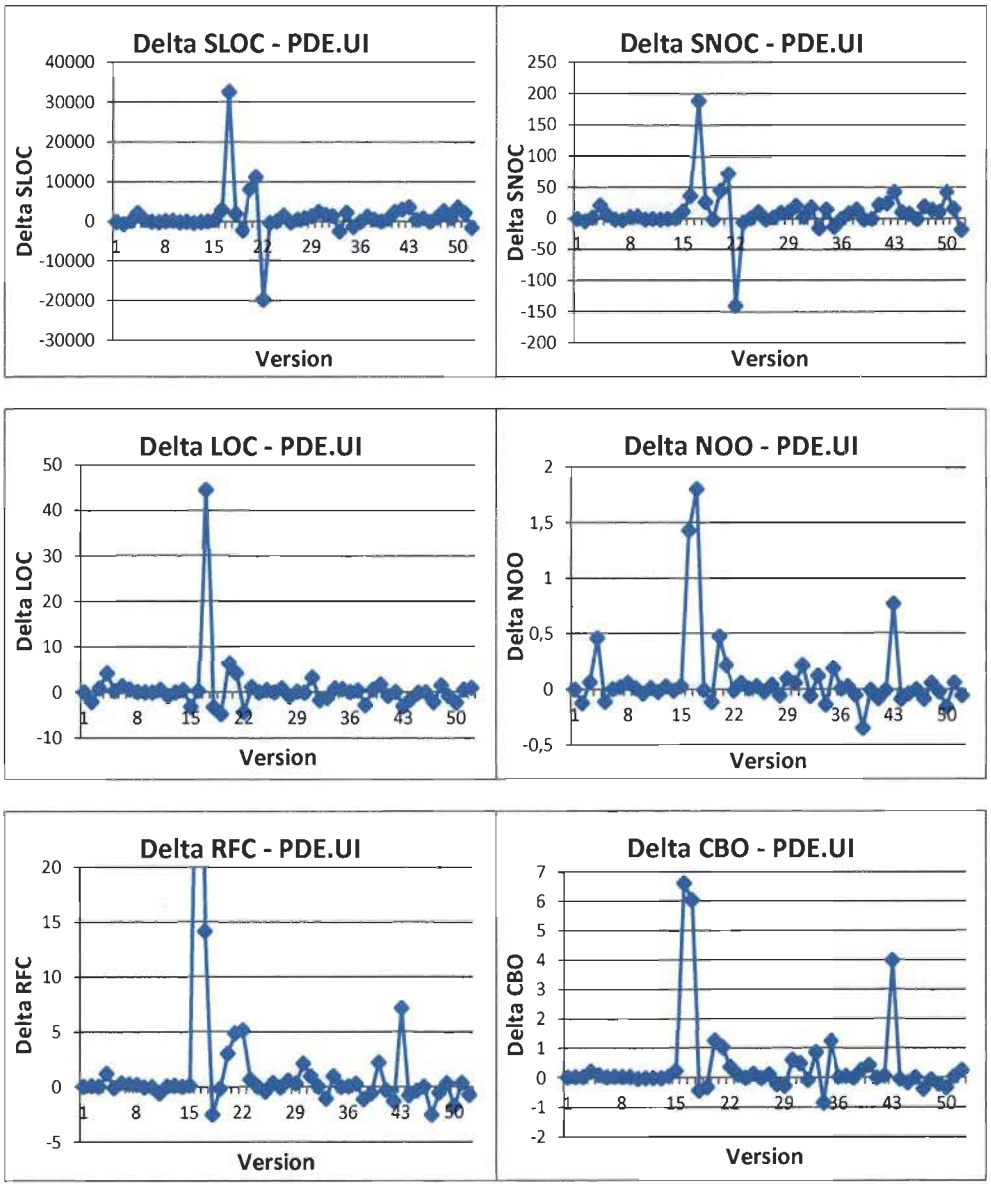


Figure 15 : Évolution des variations subies dans les métriques de taille, CBO et RFC pour PDE.UI.

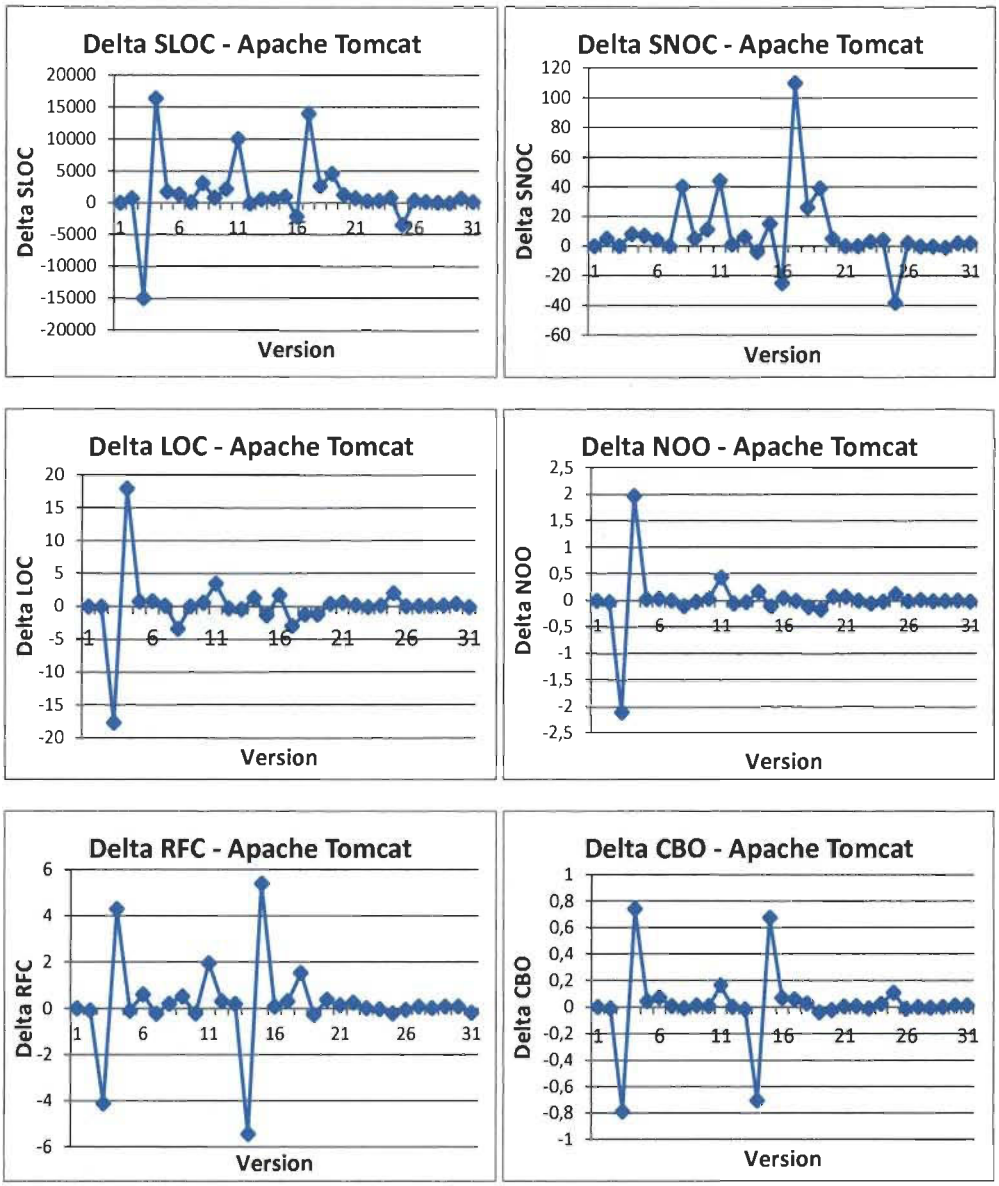


Figure 16 : *Évolution des variations subies dans les métriques de taille, CBO et RFC pour Apache Tomcat.*

Observons maintenant comment les Qi peuvent s'avérer pertinents pour étudier cette loi. Nous avons calculé les variations de Qi pour l'ensemble des itérations (valeur[itération n] = Qi[itération n]-Qi[itération n-1]). Les figures 17 à 19 présentent les résultats obtenus. L'observation de ces courbes permet de constater clairement la présence continue de changements pour deux des trois systèmes : PDE.UI et Tomcat.

Pour JDT.Debug, ces variations sont moins prononcées mais malgré tout présentes à plus petite échelle de façon continue. Les variations constatables dans les Qi indiquent donc une présence de changement de façon continue et par conséquent permettent de valider l'hypothèse 5 (1).

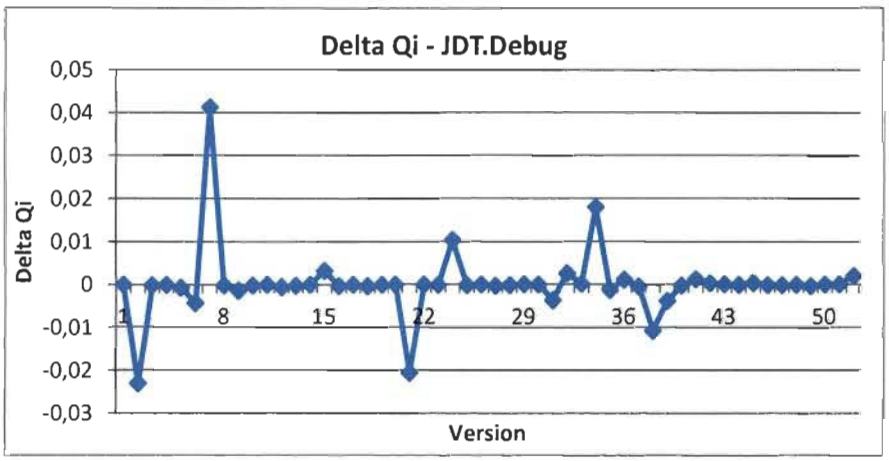


Figure 17 : Évolution des variations des Qi pour JDT.Debug.

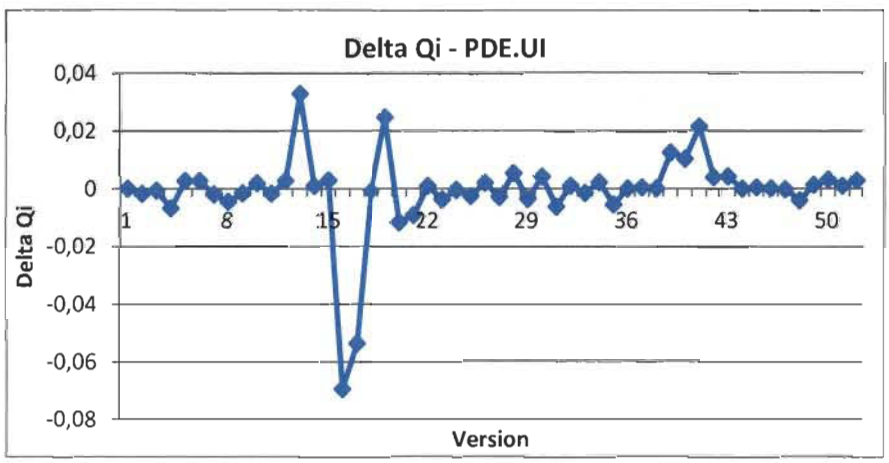


Figure 18 : Évolution des variations des Qi pour PDE.UI.

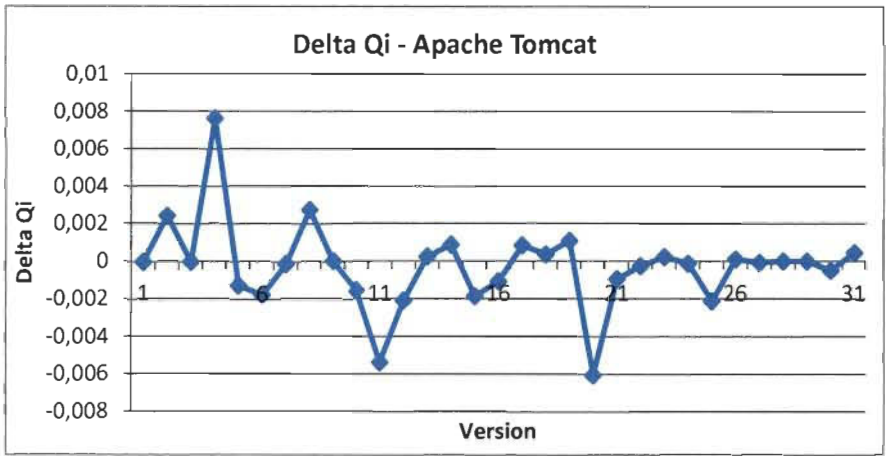


Figure 19 : Évolution des variations des Qi pour Apache Tomcat.

Ainsi, nous avons pu constater la continuité de changements à partir des métriques traditionnelles (ajouts + suppressions de classes, variations de métriques OO) et avec les Qi (variations dans les valeurs). Nous avons donc pu valider l'hypothèse 5 (1) à partir de métriques traditionnelles ainsi que par les Qi.

8.2.2 L'accroissement de la complexité (2ème loi)

La deuxième loi stipule que la complexité croît dans un système en évolution. La complexité est un attribut interne de qualité. Par conséquent, cette loi s'avère pertinente à étudier. Les études s'étant penchées sur l'applicabilité de la deuxième loi de Lehman aux systèmes issus d'un développement *open-source* (Lee, 2007; Xie, 2009) ont tous pu conclure en une croissance de la complexité. Nous prévoyons donc observer un phénomène similaire avec les systèmes à l'étude.

Comme métriques traditionnelles, nous allons nous limiter à trois métriques présentes dans la suite de Chidamber et Kemerer et qui sont liées à la complexité logicielle : les réponses pour une classe (RFC), la complexité des méthodes pondérées par classe (WMC) et le couplage entre objets (CBO). Nous pensons que les données obtenues avec ces trois métriques vont permettre d'observer une croissance dans la complexité des systèmes logiciels à l'étude.

Quant à l'utilisation des Qi dans le contexte de cette loi, ils seront analysés de deux façons. Premièrement, en étudiant l'allure générale de la courbe de ceux-ci. Deuxièmement, par l'étude des corrélations entre les Qi et les métriques de complexité précédemment mentionnées.

Hypothèse 5 (2) : La complexité d'un logiciel croît tout au long de son évolution.

Pour tenter de valider cette hypothèse, nous allons suivre la démarche suivante :

1. Illustration de la deuxième loi à partir des métriques traditionnelles
    a. Obtention de la valeur de WMC, RFC et de CBO associée à chaque itération des systèmes, représentation sous forme graphique de ces données et calcul de la pente de la courbe ainsi formée.
    b. Analyse des résultats et tentative de validation de l'hypothèse.
2. Illustration de la deuxième loi à partir des variations dans les Qi
    a. Obtention de la valeur de Qi associée à chaque itération des systèmes, représentation sous forme graphique de ces données et calcul de la pente de la courbe ainsi formée.
    b. Regroupement des corrélations entre les Qi et les différentes métriques de complexité.
    c. Analyse des résultats et tentative de validation de l'hypothèse.

Avec trois mesures logicielles traditionnelles liées à la complexité (RFC, WMC et CBO), nous avons observé l'évolution des valeurs prises pour les différentes itérations à l'étude des trois systèmes. Les figures 20 à 22 présentent les courbes obtenues à partir de ces valeurs ainsi que la pente de la droite de régression linéaire suivie par chacune de ces courbes.

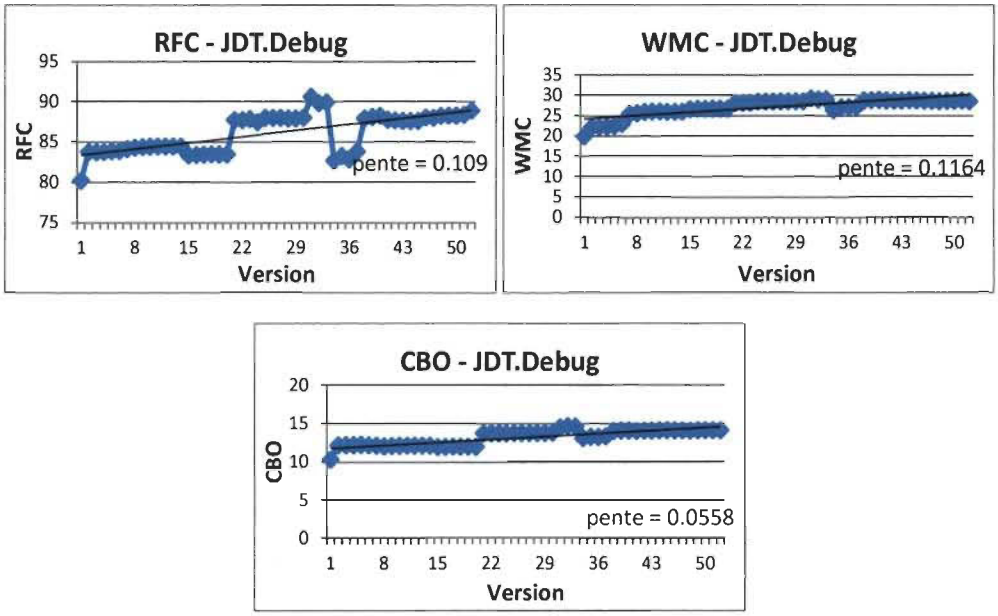


Figure 20 : Évolution des métriques de complexité pour JDT.Debug.

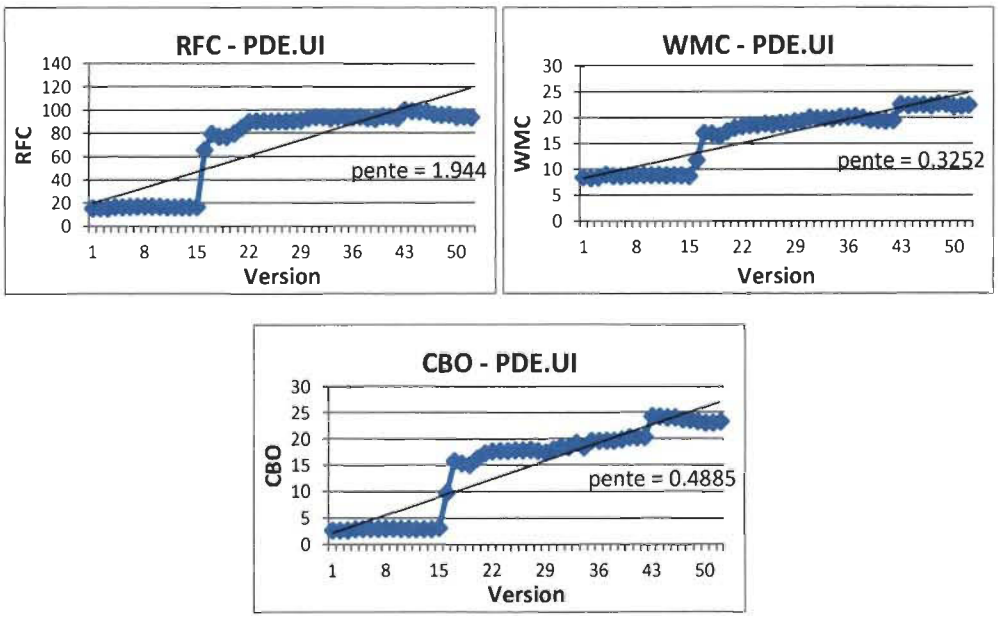


Figure 21 : Évolution des métriques de complexité pour PDE.UI.

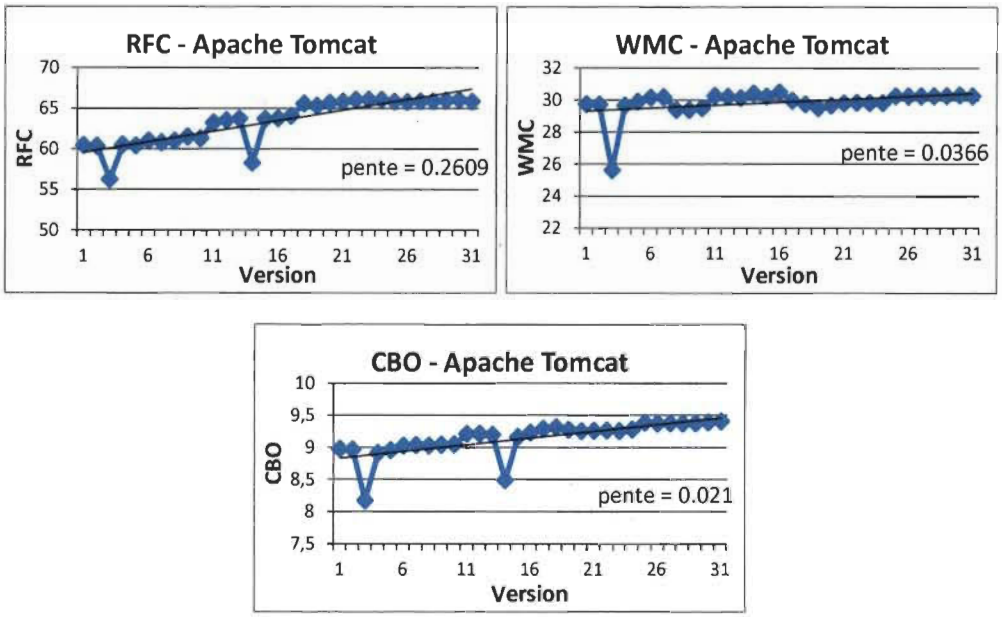


Figure 22 : Évolution des métriques de complexité pour Apache Tomcat.

Certaines baisses importantes apparaissent à quelques endroits dans les courbes. Nous pensons que ces variations sont dues à la suppression temporaire d'une dépendance importante (comme par exemple un *package* dont le code source est analysé avec le *plug-in*). Toutefois, globalement, l'ensemble des courbes offre une tendance croissante. Les pentes positives de celles-ci indiquent, en effet, une hausse généralisée de la complexité, malgré que certaines soient très près de 0. Ce qui valide donc l'hypothèse 5 (2) pour les métriques traditionnelles, c'est-à-dire que nous pouvons observer pour les trois systèmes une croissance de la complexité logicielle tout au long de leur évolution.

Nous posons notre intérêt maintenant sur les Qi. Les figures 23 à 25 présentent l'allure générale des Qi. De ces figures, nous pouvons observer que deux des trois systèmes suivent une tendance négative. Étant donné que nous avons vu précédemment que le Qi évolue de façon contraire à la plupart des métriques, nous espérons que les Qi vont évoluer négativement. Effectivement, dans deux des trois systèmes, la pente de la régression linéaire est négative. Pour le troisième système,

elle est positive mais très faible. Ces observations concordent donc avec la présence d'une complexité croissante.

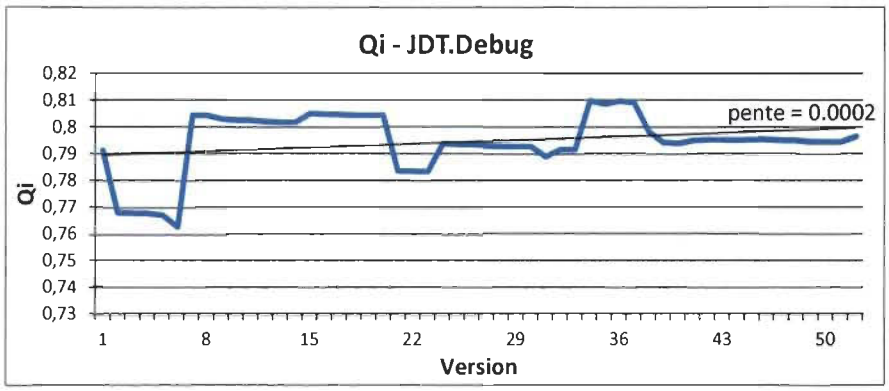


Figure 23 : Évolution des Qi pour JDT.Debug.

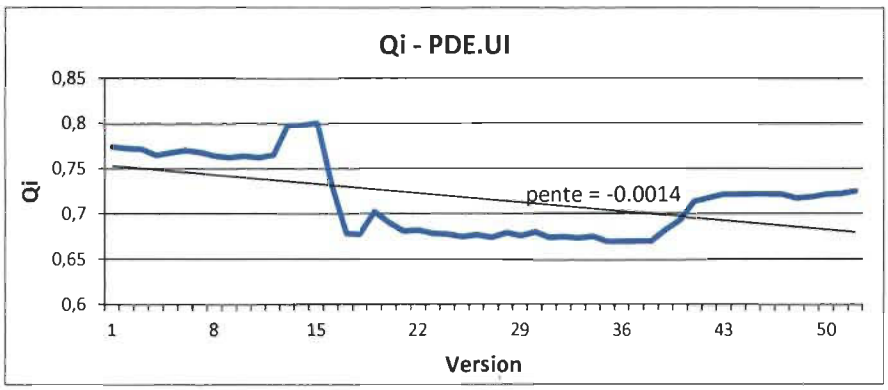


Figure 24 : Évolution des Qi pour PDE.UI.

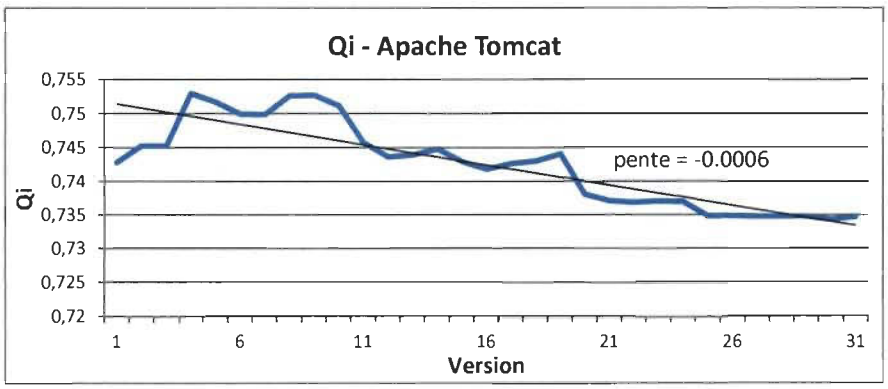


Figure 25 : Évolution des Qi pour Apache Tomcat.

Étant donné que les métriques de complexité utilisées ont pu montrer une hausse générale de la complexité, nous nous sommes aussi intéressés à la corrélation existante entre les Qi et ces métriques de complexité. Le tableau 19 présente un condensé des résultats de la section précédente en ce qui a trait aux métriques de complexité.

	RFC vs. Qi	WMC vs. Qi	CBO vs. Qi
JDT.Debug	**-0.443**	*-0.127*	**-0.373**
PDE.UI	**-0.516**	**-0.539**	**-0.475**
Apache Tomcat	**-0.830**	**-0.566**	**-0.854**

Tableau 19 : Corrélations entre le Qi et les métriques de complexité pour JDT.Debug, PDE.UI et Apache Tomcat.

Nous pouvons constater avec le tableau 19 que les corrélations sont significatives entre les Qi et les métriques de complexité. Elles sont modérées pour les deux composants d'Eclipse (JDT.Debug, PDE.UI) mais fortes (RFC et CBO) pour Apache Tomcat, le plus grand des trois systèmes. Étant donné que nous avons montré l'existence de l'accroissement de la complexité avec ces trois métriques logicielles et parce que les Qi sont corrélés à ces trois métriques, nous pouvons en conclure que les Qi permettent effectivement de bien capturer la croissance de la complexité. La tendance négative des Qi vient solidifier cette conclusion et l'hypothèse 5 (2) est donc validée.

8.2.3 L'autorégulation automatique des grands systèmes (3ème loi)

Les systèmes de grande taille ont une dynamique interne qui leur est propre. La troisième loi stipule, entre autres, que la taille d'un système de grande taille s'ajuste au cours du temps. La taille étant un attribut interne important, nous allons étudier cette loi avec les données dont nous disposons. Cette loi est toutefois peu abordée

parmi les travaux sur lesquels nous basons cette série d'expérimentations. L'approche que nous avons choisie est celle présentée par G. Xie et al. (Xie, 2009). Ces chercheurs présentent une façon simple d'observer l'auto régulation de la taille des systèmes par une analyse des variations de classes, à la recherche de fluctuations positives et négatives en alternance. Nous pensons donc qu'en procédant ainsi, nous pourrons observer un certain ajustement dans la taille.

Nous utilisons comme métrique logicielle les variations (deltas) dans plusieurs métriques de taille. L'obtention des mesures est basée sur la technique présentée dans la section 7.2.1 (première loi). À la différence, toutefois, que notre intérêt est non pas sur la continuité des variations, mais plutôt sur l'alternance de signe (positif/négatif) de celles-ci pour observer respectivement les améliorations ou dégradation de leur valeur. Nous avons aussi inclus deux métriques OO (RFC et CBO) afin de voir si une autorégulation peut aussi être constatée dans les valeurs de ces métriques.
Les Qi sont étudiés d'une façon similaire, c'est-à-dire relativement aux alternances de signes qu'ils présentent.

Hypothèse 5 (3) : Une autorégulation de la taille est observable pour les systèmes à l'étude (fluctuations positives suivies de fluctuations négatives).

Pour tenter de valider cette hypothèse, nous allons suivre la démarche suivante :

1. Illustration de la troisième loi à partir des métriques traditionnelles
    a. Calcul pour chaque métrique du delta des valeurs calculées, pour chaque itération de chaque système et représentation sous forme graphique de ces données.
    b. Analyse des courbes et tentative de validation de l'hypothèse.
2. Illustration de la troisième loi à partir des variations des Qi
    a. Calcul du delta de Qi associée à chaque itération des systèmes et représentation sous forme graphique de ces données.
    b. Analyse des courbes et tentative de validation de l'hypothèse.

Nous basons l'étude de cette loi sur les courbes obtenues lors de l'analyse de la première loi (figures 14 à 16). Pour valider notre hypothèse, nous devons observer une autorégulation dans la valeur des variations des métriques de taille retenues (SLOC, SNOC, LOC, NOO) c'est-à-dire des fluctuations positives et négatives dans l'évolution de leur valeur. Les métriques CBO et RFC sont aussi inclues dans cette étude, pour voir si ce comportement est aussi constatable pour des métriques OO non basées sur la taille.

Les figures 14 à 16 permettent effectivement de constater la présence de deltas négatifs en alternance avec de nombreux deltas positifs pour l'ensemble des métriques étudiées et ce pour les trois systèmes. Les fluctuations positives sont beaucoup plus fréquentes que celles négatives et les fluctuations négatives apparaissent de façon régulière tout au long de l'évolution des trois systèmes. Ainsi, avec ces observations, nous pouvons confirmer la présence de la troisième loi de Lehman pour l'ensemble des systèmes logiciels analysés, laquelle se traduit par une autorégulation de la taille. Ce qui valide par le fait même l'hypothèse que nous avons formulée précédemment.

Pour les Qi, des tendances similaires peuvent être observées. Encore une fois, en utilisant les figures 17 à 19 issues de l'analyse de la première loi, nous pouvons constater des variations apparaissant dans la courbe des Qi pour les trois systèmes. Contrairement aux métriques traditionnelles de taille toutefois, les variations négatives sont plus fréquentes. Elles surviennent toutefois généralement en alternance avec les variations positives, ce qui montre clairement une auto régularisation présente dans l'analyse des deltas des Qi. Ce qui valide par le fait même la présence de la troisième loi dans les systèmes à l'étude.

Les Qi permettent donc aussi d'aborder la troisième loi de Lehman. Ceux-ci permettent, comme c'est le cas aussi pour les métriques traditionnelles, d'observer des fluctuations dans la variation des mesures de façon continue. Les métriques de taille (SLOC, SNOC, LOC, NOO), les métriques OO non reliées (directement) à la

taille (CBO, RFC) tout comme les Qi sont tous pertinents pour l'étude de cette troisième loi.

8.2.4 La stabilité organisationnelle (4ème loi)

Cette loi stipule que le rythme de développement d'un système logiciel est constant et indépendant des ressources associées à un projet. Cette loi s'éloigne du concept de qualité logicielle tel qu'abordé dans ce mémoire. De plus, nous ne disposons pas de données à propos des ressources associées aux projets considérés. Par conséquent, cette loi n'a pas été retenue pour nos expérimentations.

8.2.5 La conservation de familiarité (5ème loi)

Cette loi stipule que les changements incrémentaux associés à chaque release sont relativement stables. Les recherches présentées précédemment s'étant intéressées à l'existence de cette loi pour les systèmes *open-source* n'arrivent pas à montrer une telle tendance. De plus, nous ne considérons pas que cette loi ait un effet important sur l'évolution de la qualité. Pour ces raisons, nous ne l'avons pas considérée dans notre étude.

8.2.6 La croissance continue (6ème loi)

Cette loi stipule que la taille d'un système croît de façon continue tout au long de son évolution. Tous les travaux analysés qui se sont penchés sur cette loi ont réussi à la valider pour des systèmes issus d'un développement *open-source* (Lee, 2007; Mens, 2008; Ramil, 2008; Xie, 2009). De plus, la taille est considérée comme un attribut de qualité interne important; nous allons donc nous intéresser à cette loi. Nous pensons donc que nous pourrons observer la croissance en question dans la taille.

L'étude de cette loi est étroitement liée à celle que nous avons faite de la croissance de la taille lors de l'étude de la qualité sous une perspective interne. En effet, nous y avons montré la présence d'une croissance dans la taille au travers des métriques

SNOC, SLOC, LOC et NOO pour l'ensemble des métriques logicielles de la suite CK, en plus aussi des Qi. Deux de ces métriques, le nombre de lignes de code du système et le nombre de classes, ont été respectivement utilisées par Xie (Xie, 2009) et Lee (Lee, 2007) pour valider la croissance continue. L'hypothèse à valider pour cette loi est la suivante :

Hypothèse 5 (4) : Un accroissement peut être constaté dans la taille des systèmes.

Avec la validation de l'hypothèse 2 qui a été faite précédemment, nous pouvons aussi valider l'hypothèse 5 (4) qui reprend les conclusions menées pour valider l'hypothèse dans le contexte des lois de Lehman.

8.2.7 La qualité déclinante (7ème loi)

Cette loi stipule que la qualité d'un système logiciel décroît tout au long de l'évolution de celui-ci. Parmi les études de référence qui se sont intéressées à cette loi, aucune n'est parvenue à l'observer (Lee, 2007; Xie, 2009). L'étude de Lee (Lee, 2007) a observé le phénomène inverse, c'est-à-dire une amélioration globale de la qualité. Les études sur le sujet mènent donc à des conclusions variées.

Pour étudier l'évolution de la qualité, nous procéderons de deux façons. Premièrement, par un rappel des tendances suivies par les métriques (figures 5, 6, 7, section 6.2 (évolution des métriques)). Deuxièmement, par l'observation de l'évolution des fautes au travers du temps par la recherche de patterns (selon H. Zhang et al. (Zhang, 2010)), pour en tirer des informations sur comment la qualité est contrôlée. Comme nous l'avons vu précédemment, l'apparition plus importante de fautes est un signe de moins bonne qualité.

Les Qi, par leur capacité à suivre la qualité selon une perspective interne et par le fait qu'ils parviennent à unifier de nombreux attributs logiciels internes, sont des outils pertinents pour étudier la présence de cette dynamique dans les systèmes que nous avons retenus. Ainsi, nous allons considérer la valeur des Qi associée à chaque

système comme indicateur de qualité, et ainsi étudier la courbe formée par les valeurs prises par cette métrique pour les différents systèmes au cours du temps.

Hypothèse 5 (5) : La qualité des systèmes logiciels se dégrade au cours du temps.

Pour tenter de valider cette hypothèse, nous allons suivre la démarche suivante :

1. Illustration de la septième loi à partir de métriques traditionnelles
    a. Obtention du nombre de fautes associé à chacune des itérations du système et représentation sous forme graphique de ces données.
    b. Analyse des courbes obtenues avec les patterns (Zhang, 2010) et tentative de validation de l'hypothèse.
2. Illustration de la septième loi à partir des valeurs des Qi
    a. Obtention du Qi associé à chacune des itérations du système et représentation sous forme graphique de ces données.
    b. Analyse du graphique formé ainsi que de la pente de celle-ci et tentative de validation de l'hypothèse.

Tout d'abord, nous avons vu dans la section 6.2 (évolution des métriques) que les valeurs des métriques, de façon générale, augmentent avec le temps. Nous avons aussi constaté que les Qi capturent bien l'information procurée par ces métriques.
Les figures 26, 27 et 28 présentent l'évolution du nombre de fautes répertoriées pour chacune des itérations des trois systèmes. H. Zhang (Zhang, 2010) a étudié l'évolution des fautes avec l'aide de patterns récurrents pour deux des trois systèmes que nous utilisons : JDT.Debug et PDE.UI.

Pour JDT.Debug, il a été défini comme étant de bonne qualité globalement en raison du nombre décroissant de fautes au cours du temps. Nous pouvons, en effet, constater que l'amplitude des pics de fautes diminue au cours du temps. Pour ce système, la qualité est donc en contrôle.

Le système PDE.UI est qualifié comme présentant un pattern de montagnes russes, c'est-à-dire que sa qualité n'est pas sous contrôle. Les amplitudes sont au plus fort au centre de la période d'évolution étudiée (itérations 21 et 33). À l'exception de ces deux périodes, la qualité semble être beaucoup plus sous contrôle.

Quant à Tomcat, ce projet ne fait pas parti de ceux analysés par H. Zhang. Toutefois, à partir des patterns qu'il a énoncés, nous pouvons affirmer que celui-ci suit aussi un pattern de montagnes russes en raison des nombreux pics visibles dans la courbe correspondante. La qualité semble toutefois être beaucoup plus en contrôle vers la fin de cette période avec une stabilisation du nombre de fautes.

L'accroissement des valeurs des métriques nous pousse donc à croire en une détérioration de la qualité pour les trois systèmes à l'étude. L'étude des fautes vient corroborer pour PDE.UI et Apache Tomcat, mais l'étude reste assez subjective.

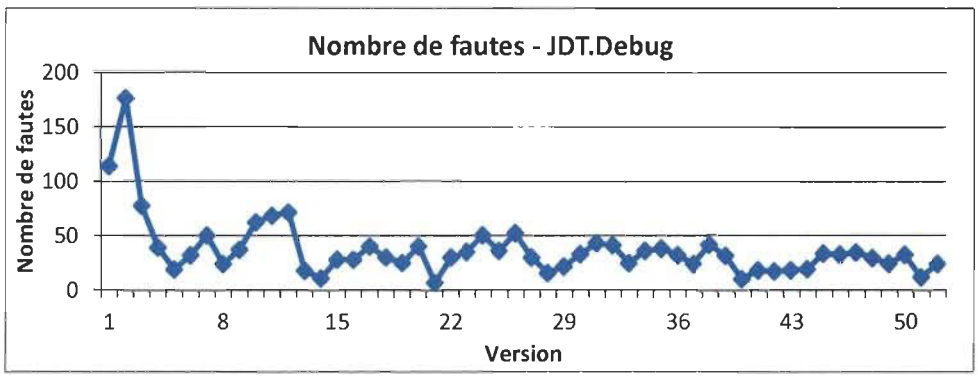


Figure 26 : Évolution du nombre de fautes pour JDT.Debug.

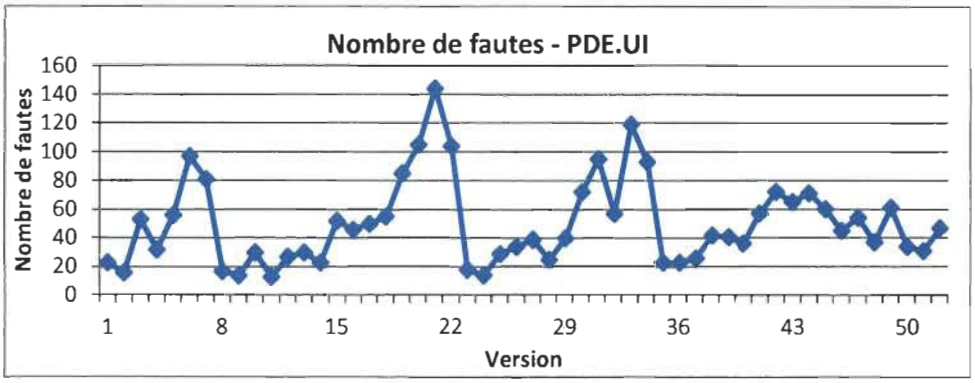


Figure 27 : Évolution du nombre de fautes pour PDE.UI.

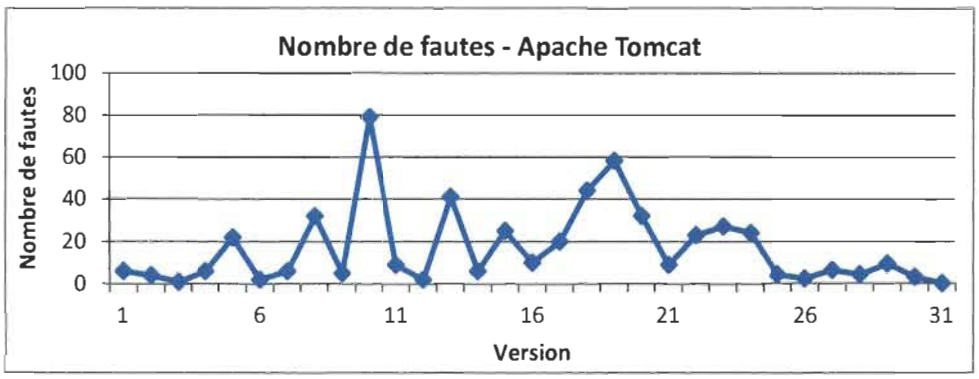


Figure 28 : Évolution du nombre de fautes pour Apache Tomcat.

Pour étudier l'information que peuvent apporter les Qi dans le contexte de cette loi, nous allons tout d'abord étudier la pente de la courbe de régression pour l'évolution des Qi des trois systèmes. Nous allons ensuite voir comment les Qi se comportent comparativement à l'évolution des fautes. Rappelons qu'une forte valeur de Qi est signe d'une bonne qualité, et vice-versa. Les figures 29 à 31 présentent la courbe formée par les Qi obtenus à partir de chaque capture de système, superposée avec la fréquence de fautes répertoriées. La courbe de régression linéaire est affichée pour chacune des courbes de Qi avec la valeur de la pente correspondante, et les valeurs sur l'axe des y sont normalisées entre 0 et 1 (min-max).

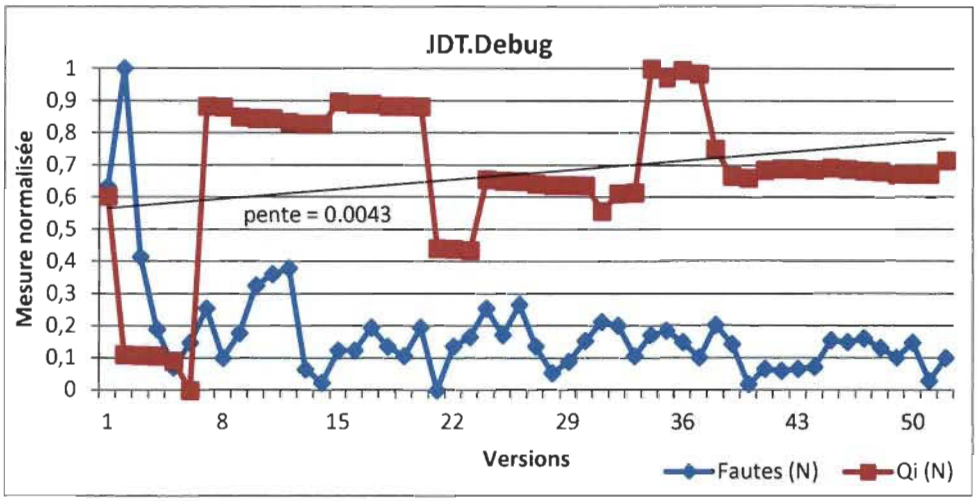


Figure 29 : Évolution du nombre de fautes superposé aux Qi pour JDT.Debug.

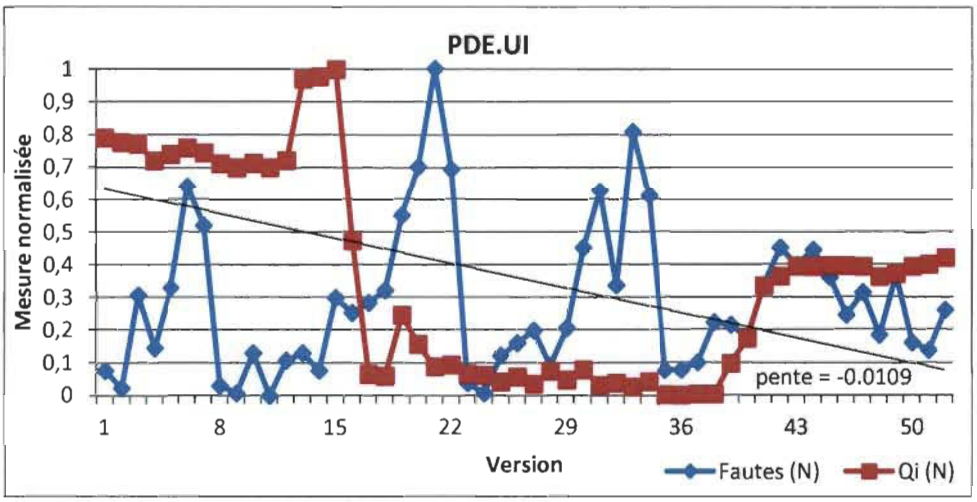


Figure 30 : Évolution du nombre de fautes superposé aux Qi pour PDE.UI.

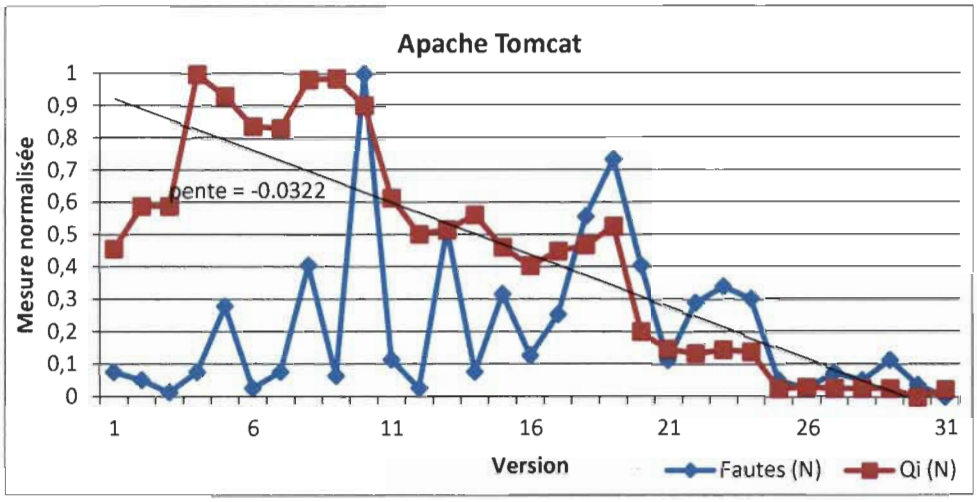


Figure 31 : Courbes obtenues avec les Qi associés aux systèmes à l'étude.

L'analyse de la courbe de JDT.Debug (corroborée par le calcul de la pente) permet de constater une stabilité dans la progression des Qi. En effet, la courbe de régression est très près de l'horizontal et l'évolution se fait par paliers semblant alterner autour d'une ligne médiane. Ce qui correspond à la constatation faite par H. Zhang d'un contrôle de la qualité. De plus, l'itération où le nombre de fautes est le plus élevé (itération 2) correspond à une baisse prononcée des Qi. Les Qi semblent donc ici bien capter l'évolution de la qualité.

PDE.UI montre aussi une tendance négative dans les Qi, causée principalement par la baisse importante entre les itérations 15 et 17. S'en suit par la suite un long palier où les Qi sont à leur plus bas, s'étendant entre les itérations 17 et 38. Pendant cette période, le nombre de fautes est hors de contrôle, et deux pics importants apparaissent. Une fois cette période passée, les Qi remontent. Globalement, les Qi indiquent donc la perte de contrôle de la qualité.

Apache Tomcat présente une tendance décroissante de façon régulière des Qi. Ce qui indique une baisse prononcée de la qualité. Nous avons identifié pour ce système un pattern de montagnes russes dans les fautes, donc une perte de contrôle de la qualité. Les Qi permettent donc ici aussi d'identifier un tel pattern.

Avec les observations que nous avons effectuées, les Qi peuvent-ils permettre d'affirmer que la décroissance de la qualité est constatable sur les systèmes dont nous disposons? Les données recueillies de PDE.UI et Tomcat vont en ce sens. Par contre, ce n'est pas le cas pour JDT.Debug où une stabilité est identifiée. Il ne faut, toutefois, pas oublier que la septième loi parle de dégradation dans le cas où aucune opération de maintenance n'est effectuée pour la stabiliser. Par conséquent, JDT.Debug ne permet pas d'infirmer la septième loi. Nous pouvons donc affirmer que les Qi nous permettent de valider notre hypothèse 5 (5) et que nous assistons avec nos données à une dégradation de la qualité.

8.2.8 Les mécanismes de feedback (8ème loi)

Cette loi stipule que les systèmes logiciels en évolution doivent implémenter des mécanismes de feedback. Étant donné que nous ne disposons pas de mesures logicielles relatives aux mécanismes de feedback et parce que cette loi ne concerne pas directement la problématique d'évolution de la qualité, nous n'avons pas retenu cette loi pour nos expérimentations.

8.3 Conclusion sur les lois de Lehman

Cinq des huit lois de Lehman ont été retenues pour nos expérimentations : la première loi (le changement continu), la deuxième loi (la complexité croissante), la troisième loi (l'auto régulation des systèmes de grande taille), la sixième loi (la croissance continue) et la septième loi (la qualité déclinante).

Les cinq lois analysées ont pu être validées. Le but de cette section était de présenter plusieurs dynamiques d'évolution inévitables pouvant influer sur l'allure suivie par la qualité logicielle, et de placer les Qi comme des outils valables pour l'étude de ces dynamiques d'évolution. Ces objectifs nous semblent donc remplis.

CHAPITRE 9
CONCLUSION

La maintenance logicielle représente un coût important dans le développement d'un logiciel. Par conséquent, voir à réduire les coûts de la maintenance peut s'avérer une solution pertinente pour réduire les coûts globaux du développement. Un logiciel de qualité sera plus facilement maintenable. En effet, si sa qualité se dégrade soudainement en raison de mauvaises décisions de design, il risque de devenir plus difficile ce qui inévitablement fera grimper les coûts de développement. Mieux comprendre l'évolution de la qualité d'un système permet donc d'en prévoir le comportement et d'aider à la contrôler.

Nous avons tout d'abord présenté la problématique d'évolution logicielle. Nous avons pu constater que de considérer ce phénomène est inévitable et essentiel pour le développement de logiciels. Nous avons présenté les lois de Lehman, en exposant quelques dynamiques importantes apparaissant dans les systèmes logiciels propriétaires et nous avons exposé plusieurs travaux s'étant intéressé à leur applicabilité aux systèmes *open-source*.

Nous avons ensuite présenté le concept de qualité logicielle. Nous avons vu que les attributs de qualité permettent d'illustrer plusieurs facettes de la qualité d'un logiciel, tant d'une perspective interne qu'externe. Nous avons aussi exposé comment l'étude des attributs de qualité peut être appliquée dans un contexte d'évolution logicielle.

Troisièmement, nous avons exposé le modèle des indicateurs de qualité (Qi), c'est-à-dire comment ils sont calculés et quelles propriétés leur a été associées (dont ses capacités à unifier plusieurs attributs logiciels internes).

La pertinence d'utilisation des métriques logicielles pour suivre l'évolution de la qualité a été évaluée sous trois perspectives distinctes. Par l'utilisation des métriques formant la suite de Chidamber et Kemerer (CBO, LCOM, WMC, RFC, DOIH) et la

taille (LOC, NOC), de nombreux attributs internes de qualité ont été considérés : la complexité, le couplage, la taille, la cohésion et l'héritage. Tout d'abord, nous avons traité la qualité selon une perspective interne, c'est-à-dire en se basant sur les propriétés structurelles du code. Des relations ont pu être démontrées entre les Qi et de nombreux attributs reliés à la qualité. De plus, les relations avec la taille du système ont permis de situer les Qi comme des indicateurs comparables à ceux davantage reconnus. Ces deux étapes nous ont permis d'obtenir des résultats significatifs. Avec une perception externe au système (perspective externe), nous avons considéré la qualité relativement au nombre de fautes par version. Nous avons vu que l'étude des changements (ajouts/suppressions) permet de capturer de l'information relative aux versions pour lesquelles davantage de fautes sont trouvées. Enfin, nous avons étudié comment les Qi peuvent permettre d'illustrer plusieurs lois de Lehman, c'est-à-dire de nombreuses dynamiques inhérentes à l'évolution logicielle (lois de Lehman). L'ensemble des lois étudiées (cinq) ont pu être validées.

Nos études se sont concentrées essentiellement au niveau système. Nous ne sommes pas entrés en profondeur dans l'évolution des classes formant ceux-ci, ce qui aurait pu être une étude intéressante en soit. Nous avons donc laissé ce travail colossal de côté dans notre étude en raison de manque de données et de temps, dû à la fréquence de versions analysées au niveau système. Mais une telle étude pourrait apporter de l'information très pertinente sur le comportement d'une classe logicielle en évolution.

Les avenues d'extension avec une problématique telle que celle exposée dans ce mémoire sont abondantes. L'évolution logicielle est un domaine de recherche en plein essor, en raison principalement de la disponibilité grandissante de données permettant de procéder à de telles études. Dans le contexte particulier de la compréhension de l'évolution d'attributs de qualité, qui est aujourd'hui un enjeu important en raison, entre autres, de la taille des systèmes actuels, nos recherches ont pu présenter certaines métriques orientées objet et en particulier les indicateurs de qualité (Qi) comme des outils pertinents pour procéder à des investigations sur l'évolution de la qualité. Une telle métrique unificatrice de plusieurs attributs qualité,

dans un contexte d'évolution, faciliterait grandement le travail d'analyse rétrospective pour l'assurance qualité en simplifiant le nombre de métriques à (collecter et à) analyser.

BIBLIOGRAPHIE

[Abran 10] Abran A. (2010). *Software Metrics and Software Metrology.* John Wiley & Sons Interscience and IEEE-CS Press.

[Aggarwal 06] Aggarwal K.K., Singh Y., Kaur A., Malhotra R. (2006). *Empirical Study of Object-Oriented Metrics.* In Journal of Object-Technology, vol. 5, no. 8, pp. 149-173.

[Aggarwal 07] Aggarwal K.K., Singh Y., Kaur A., Malhotra R. (2007). *Investigating effect of design metrics on fault proneness in object-oriented systems.* In Journal of Object Technology, vol. 6, no. 10, pp. 127-141

[Al Ajlan 09] Al-Ajlan A. (2009). *The Evolution of Open Source Software Using Eclipse Metrics.* In International Conference on New Trends in Information and Service Science 2009, pp. 211-218.

[Ali 09] Ali S., Maqbool O. (2009). *Monitoring software evolution using multiple types of changes.* In International Conference on Emerging Technologies 2009, pp. 410-415

[Ambu 06] Ambu W., Concas G., Marchesi M., Pinna S. (2006). *Studying the evolution of quality metrics in an agile/distributed project.* In Extreme Programming and Agile Processes in Software Engineering 2006, pp. 85-93

[Badri 09] Badri M., Badri L., Touré F. (2009). *Empirical Analysis of Object-Oriented Design Metrics: Towards a New Metric Using Control Flow Paths and Probabilities.* In Journal of Object Technology, vol.8, no.6, pp. 123-142.

[Badri 11] Badri M., Touré F. (2011). *Empirical Analysis for Investigating the Effect of Control Flow Dependencies on Testability of Classes.* In 23rd International Conference on Software

Engineering and Knowledge Engineering (SEKE), USA.

[Badri 12] Badri M., Toure F. (2012). *Evaluating the Effect of Control Flow on the Unit Testing Effort of Classes: An Empirical Analysis*. In Advances in Software Engineering, vol. 2012, Article ID 964064, 13 pages.

[Bansiya 02] Bansiya J., Davis C. (2002). *A Hierarchical Model for Object-Oriented Design Quality Assessment.* In IEEE Transactions Software Eng., vol. 28, no. 1, pp. 4-17.

[Basili 92] Basili V. (1992). *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. In Technical Report, CS-TR-2956, University of Maryland, September 1992.

[Basili 96] Basili V., Briand L., Melo W.L. (1996). *A Validation of Object Oriented Design Metrics as Quality Indicators*. In IEEE Transactions Software Eng. 1996.

[Briand 00] Briand L.C., Wüst J., Daly J.W., Porter D.V. (2000). *Exploring the relationships between design measures and software quality in object-oriented systems*. In Journal of Systems and Software 51, pp. 245-273.

[Chidamber 94] Chidamber S.R., Kemerer C.F. (1994). *A Metrics Suite for Object Oriented Design*. In IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493.

[Chidamber 98] Chidamber S.R., Darcy D.P., Kemerer C.F. (1998). *Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis*. In IEEE Transactions on Software Engineering, vol. 24, no. 8, pp. 629-639.

[Dagpinar 03] Dagpinar M., Jahnke J.H. (2003). *Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison*. In Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03), pp. 155-164.

[El Emam 01] El Emam K., Benlarbi S., Goel N., Rai S.N. (2001). *The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics*. In IEEE Transactions on Software Engineering, vol. 27, no. 7, pp. 630-650.

[Erlikh 00] Erlikh L. (2000). *Leveraging Legacy System Dollars for E-Business*. In IT Pro, vol. May/June 2000, pp. 17-23.

[Eski 11] Eski S., Buzluca F. (2011). *An Empirical Study on Object-Oriented Metrics and Software Evolution in Order to Reduce Testing Costs by Predicting Change-Prone Classes*. In 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 566-571.

[Fenton 95] N Fenton N.E., Whitty R. (1995). *Software Quality Assurance and Measurement: A Worldwide Perspective*. London: International Thomson Computer Press, page 7.

[Godfrey 01] Godfrey M., Tu Q. (2001). *Growth, Evolution and Structural Change in Open Source Software*. In Proceedings of the 4th International Workshop on Principles of Software Evolution, pp. 103-106.

[Godfrey 08] Godfrey M.W., German D.M. (2008). *The Past, Present, and Future of Software Evolution*. In Frontiers of Software Maintenance 2008, pp. 129-138

[van Gurp 02] van Gurp J., Bosch J. (2002). *Design Erosion: Problems & Causes*. In Journal of Systems and Software, vol. 61, no. 2, pp. 105-119.

[Herraiz 09] Herraiz I. (2009). *A Statistical Examination of the Evolution and Properties of Libre Software*. In IEEE International Conference on Software Maintenance (ICSM '09), pp. 439-442.

[Honglei 09] Honglei T., Wei S., Yanan Z. (2009). *The Research on Software Metrics and Software Complexity Metrics*. In 2009 International

Forum on Computer Science-Technology and Applications, pp. 131-136.

[Izurieta 06] Izurieta C., Bieman J. (2006). *The Evolution of FreeBSD and Linux*. In Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE '06), pp. 204-211.

[Jermakovics 07] Jermakovics A., Scotto M., Succi G. (2007). *Visual Identification of Software Evolution Patterns*. In 9th International Workshop on Principles of Software Evolution (IWPSE '07): in Conjunction with the 6th ESEC/FSE Joint Meeting, pp. 27-30

[Kemerer 99] Kemerer C.F., Slaughter S. (1999). *An empirical approach to studying software evolution*. In IEEE Transactions on Software Engineering, vol. 25, no. 4, pp. 493-509.

[Lee 07] Lee Y., Yang J., Chang K.H. (2007). *Metrics and evolution in open source software*. In 7th International Conference on Quality Software (QSIC '07), pp. 191-197.

[Lehman 97] Lehman M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski W.M. (1997). *Metrics and laws of software evolution-the nineties view*. In Proceedings of the Fourth International Software Metrics Symposium, pp. 20-32.

[Lehman 03] Lehman M.M., Fernandez-Ramil J. (2003). *Software Evolution – Background, Theory, Practice*. In Information Processing Letters, vol. 88, no.1-2, pp. 33-44.

[Mens 08a] Mens T., Fernandez-Ramil J., Degrandsart S. (2008a). *The Evolution of Eclipse*. In IEEE International Conference on Software Maintenance 2008 (ICSM '08), pp. 386-395

[Mens 08b] Mens T., Demeyer S. (2008b). *Software Evolution*. Springer.

[Murgia 09] Murgia A., Concas G., Pinna S., Tonelli R., Turnu I. (2009). *Empirical Study of Software Quality Evolution in Open Source Projects Using Agile Practices*. In CoRR, Vol. abs/0905.3287.

[Parnas 94] Parnas D.L. (1994). *Software aging*. In Proceedings of the 16th International Conference on Software Engineering (ICSE '94), pp. 279-287.

[Ramil 08] Fernandez-Ramil J., Lozano A., Wermelinger M., Capiluppi A. (2008). *Empirical studies of Open-Source Evolution*. In Mens Tom and Demeyer, Serge eds. Software Evolution. Berlin:Springer, pp. 263-288.

[Reddy 09] Reddy K.N., Rao A.A. (2009). *A Quantitative Evaluation of Software Quality Enhancement by Refactoring Using Dependency Oriented Complexity Metrics*. In 2009 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET '09), pp. 1011-1018.

[Sharma 02] Sharma S., Sugumaram V., Rajagopalan B. (2002). *A Framework for Creating Hybrid-Open Source Software Communities*. In Information Systems Journal, vol. 12, no. 1, pp. 7-25.

[Singh 10] Singh Y., Kaur A., Malhotra R. (2010). *Empirical validation of object-oriented metrics for predicting fault proneness models*. In Software Quality Journal, vol. 18, no. 1, pp. 3-35.

[Sommerville 07] Sommerville I. (2007). *Software Engineering*. 8th Edition, Addison Wesley.

[Subramanyam 03] Subramanyan R, Krishnan M.S. (2003). *Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects*. In IEEE Transactions on Software Engineering, vol. 29, no. 4, pp. 297-310.

[Tang 99] Tang M.-H., Kao M.-H., Chen M.H. (1999). *An Empirical Study*

on Object-Oriented Metrics. In Proceedings of Sixth International Software Metrics Symposium, pp. 242-249.

[Toure 07] Toure F. (2007). *Indicateur de qualité pour les systèmes orientés objet : Vers un modèle unifiant plusieurs métriques*. Mémoire. Trois-Rivières, Université du Québec à Trois-Rivières.

[Voas 04] Voas J., Agresti W.W. (2004). *Software Quality from a Behavioral Perspective*. In IT Professional, vol. 6, no. 4, pp. 46-50.

[Xie 09] Xie G., Chen J., Neamtiu I. (2009). *Towards a better understanding of software evolution: An Empirical study on open source software*. In ICSM '09, pp. 51-60.

[Yu 11] Yu L., Ramaswamy S., Nail A. (2011). *Using Bug Reports As a Software Quality Measure*. In Proceedings of the 16th International Conference on Information Quality (ICIQ '11), pp. 277-286.

[Zhang 09] H. Zhang (2009). *An investigation of the relationships between lines of code and defects*. In ICSM '09, pp. 274-283.

[Zhang 10] Zhang H, Kim S. (2010). *Monitoring software quality evolution for defects*. In IEEE Software, vol. 27, no. 4, pp. 58-64.

[Zhou 06] Zhou Y., Leung H. (2006). *Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults*. In IEEE Transactions on Software Engineering, vol. 32, no. 10, pp. 771-789.

ANNEXE 1

RÉSULTATS DE L'ANALYSE PAR VERSION

Annexe 1a-1 : Caractéristiques des captures pour JDT.Debug au niveau du système

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
1	2002-03-28	12201	127	114
2	2002-04-30	11689	109	176
3	2002-05-31	11710	109	77
4	2002-06-26	11757	109	39
5	2002-07-29	11830	109	19
6	2002-08-26	12426	110	32
7	2002-09-30	15371	115	50
8	2002-10-29	15396	115	24
9	2002-11-25	15647	116	37
10	2002-12-16	15686	116	62

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
11	2003-01-27	15693	116	68
12	2003-02-26	15604	116	71
13	2003-03-27	15608	116	18
14	2003-04-28	15609	116	11
15	2003-05-26	16705	120	28
16	2003-06-30	16739	120	28
17	2003-07-28	16751	120	40
18	2003-09-01	16761	120	30
19	2003-09-29	16765	120	25
20	2003-10-27	16761	120	40

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
21	2003-12-01	28283	198	7
22	2003-12-16	28317	198	30
23	2004-01-26	28349	198	35
24	2004-03-01	28976	200	50
25	2004-03-29	29077	201	36
26	2004-04-26	29085	201	52
27	2004-05-28	29125	201	30
28	2004-06-24	29146	201	16
29	2004-07-26	29084	201	22
30	2004-08-30	29086	201	33
31	2004-09-27	28912	196	43
32	2004-11-01	29250	200	41
33	2004-11-22	29276	200	25
34	2004-12-16	29836	224	36
35	2005-01-31	30509	224	38
36	2005-02-28	30519	223	32

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
37	2005-04-01	30445	222	24
38	2005-04-25	30451	210	41
39	2005-05-25	30823	210	31
40	2005-06-23	30874	210	10
41	2005-07-25	30849	211	18
42	2005-08-22	30959	212	17
43	2005-09-26	30958	212	18
44	2005-10-28	31171	213	19
45	2005-11-28	31154	213	33
46	2005-12-14	31094	213	32
47	2006-01-30	31256	214	34
48	2006-02-27	31064	213	29
49	2006-03-29	31137	213	24
50	2006-04-27	31142	213	32
51	2006-06-01	31163	213	12
52	2006-06-26	31884	218	24

Annexe 1a-2 : Caractéristiques des captures pour JDT.Debug au niveau des classes

# Itération	Date	Valeurs moyennes des métriques OO pour les classes								
		Qi	LOC	CBO	RFC	LCOM	NOC	WMC	DIT	NOO
1	2002-03-28	0.791	96.1	10.3	80.2	183.7	2.70	20.0	2.64	10.2
2	2002-04-30	0.768	107.2	12.2	83.8	219.5	1.79	22.3	2.48	11.0
3	2002-05-31	0.768	107.4	12.2	83.8	219.5	1.79	22.3	2.48	11.0
4	2002-06-26	0.768	107.9	12.2	83.8	223.6	1.79	22.4	2.48	11.0
5	2002-07-29	0.767	108.5	12.2	83.9	223.6	1.79	22.6	2.48	11.0
6	2002-08-26	0.763	113.0	12.2	83.9	219.7	1.77	23.1	2.46	11.0
7	2002-09-30	0.804	133.7	12.1	84.2	311.6	1.70	25.5	2.45	12.3
8	2002-10-29	0.804	133.9	12.0	84.3	319.4	1.70	25.7	2.45	12.4
9	2002-11-25	0.803	134.9	12.1	84.4	320.1	2.18	26.0	2.44	12.7
10	2002-12-16	0.803	135.2	12.1	84.5	322.7	2.18	26.1	2.44	12.7
11	2003-01-27	0.803	135.3	12.1	84.5	321.9	2.18	26.1	2.44	12.7
12	2003-02-26	0.802	134.5	12.1	84.5	319.3	2.18	26.1	2.44	12.7
13	2003-03-27	0.802	134.6	12.1	84.5	319.3	2.18	26.1	2.44	12.7
14	2003-04-28	0.802	134.6	12.1	84.5	319.3	2.18	26.1	2.44	12.7
15	2003-05-26	0.805	139.2	12.0	83.4	317.3	2.11	26.7	2.39	12.7
16	2003-06-30	0.805	139.5	12.0	83.4	317.2	2.11	26.8	2.39	12.7
17	2003-07-28	0.805	139.6	12.0	83.5	316.7	2.11	26.8	2.39	12.7
18	2003-09-01	0.804	139.7	12.0	83.5	316.5	2.11	26.8	2.39	12.7

19	2003-09-29	0.804	139.7	12.0	83.5	316.5	2.11	26.8	2.39	12.7
20	2003-10-27	0.804	139.7	12.0	83.5	316.5	2.11	26.8	2.39	12.7
21	2003-12-01	0.784	142.8	13.7	87.8	320.0	2.11	28.2	2.47	13.4
22	2003-12-16	0.784	143.0	13.7	87.8	319.9	2.11	28.2	2.48	13.4
23	2004-01-26	0.783	143.2	13.7	87.9	320.7	2.11	28.3	2.48	13.4
24	2004-03-01	0.794	144.9	13.7	87.5	326.9	2.09	28.6	2.47	13.4
25	2004-03-29	0.794	144.7	13.7	88.0	326.4	2.08	28.5	2.47	13.4
26	2004-04-26	0.794	144.7	13.8	88.0	326.5	2.08	28.6	2.47	13.4
27	2004-05-28	0.793	144.9	13.8	88.1	326.7	2.08	28.6	2.47	13.4
28	2004-06-24	0.793	145.0	13.8	88.0	325.3	2.08	28.6	2.47	13.4
29	2004-07-26	0.793	144.7	13.8	88.0	325.3	2.08	28.6	2.47	13.4
30	2004-08-30	0.793	144.7	13.8	88.1	324.6	2.08	28.6	2.47	13.4
31	2004-09-27	0.789	147.5	14.5	90.7	369.3	2.13	29.4	2.51	14.1
32	2004-11-01	0.792	146.3	14.7	89.9	364.9	2.07	29.1	2.49	14.0
33	2004-11-22	0.792	146.4	14.6	90.0	364.7	2.07	29.1	2.49	14.0
34	2004-12-16	0.810	133.2	13.1	82.7	326.1	1.84	26.5	2.60	12.6
35	2005-01-31	0.809	136.2	13.3	83.3	342.8	1.86	27.1	2.60	12.8
36	2005-02-28	0.810	136.9	13.3	82.9	338.0	1.80	27.2	2.52	12.7
37	2005-04-01	0.809	137.1	13.3	83.8	339.3	1.81	27.2	2.66	12.7
38	2005-04-25	0.798	145.0	14.0	88.0	358.9	1.91	28.8	2.78	13.4
39	2005-05-25	0.794	146.8	14.1	88.1	359.3	1.91	28.9	2.78	13.5
40	2005-06-23	0.794	147.0	14.1	88.3	360.5	1.91	29.0	2.78	13.5
41	2005-07-25	0.795	146.2	14.0	87.8	358.5	1.91	28.8	2.78	13.4

42	2005-08-22	0.795	146.0	14.1	87.7	357.0	1.91	28.8	2.77	13.4
43	2005-09-26	0.795	146.0	14.1	87.7	357.0	1.91	28.8	2.77	13.4
44	2005-10-28	0.795	146.3	14.1	87.6	355.6	1.90	28.9	2.77	13.4
45	2005-11-28	0.796	146.3	14.1	87.7	355.3	1.90	28.9	2.77	13.4
46	2005-12-14	0.795	146.0	14.1	88.1	355.5	1.90	28.8	2.80	13.4
47	2006-01-30	0.795	146.1	14.1	88.0	356.7	1.89	28.8	2.80	13.4
48	2006-02-27	0.795	145.8	14.1	88.3	358.2	1.90	28.7	2.81	13.4
49	2006-03-29	0.795	146.2	14.1	88.3	358.4	1.90	28.8	2.81	13.4
50	2006-04-27	0.795	146.2	14.1	88.3	358.4	1.90	28.8	2.81	13.4
51	2006-06-01	0.795	146.3	14.1	88.4	359.1	1.90	28.8	2.81	13.4
52	2006-06-26	0.796	146.3	14.1	88.9	357.7	1.95	28.7	2.83	13.3

Annexe 1b-1 : Caractéristiques des captures pour PDE.UI au niveau du système

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
1	2002-08-29	9519	121	23
2	2002-10-01	8961	117	16
3	2002-10-29	9214	119	53
4	2002-11-26	11493	141	32
5	2002-12-17	12022	147	56
6	2003-01-29	12129	146	97
7	2003-02-27	12050	144	81
8	2003-03-27	12309	147	17
9	2003-04-29	12639	151	14
10	2003-05-27	12555	150	30
11	2003-07-01	12631	150	13
12	2003-07-29	12423	149	27
13	2003-08-28	12414	149	30
14	2003-09-30	12564	150	23
15	2003-10-29	13051	162	52
16	2003-11-25	16103	199	46

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
17	2003-12-18	48670	388	50
18	2004-01-30	50701	415	55
19	2004-02-26	48612	414	85
20	2004-03-30	56841	459	105
21	2004-04-28	68101	532	144
22	2004-05-28	48491	392	104
23	2004-06-24	48184	386	18
24	2004-07-27	48296	387	14
25	2004-08-19	49850	398	29
26	2004-09-23	49713	397	34
27	2004-11-01	50300	399	39
28	2004-11-30	51207	409	25
29	2004-12-21	52641	420	40
30	2005-02-01	55228	441	72
31	2005-03-01	57030	444	95
32	2005-03-31	58544	462	57

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
33	2005-04-26	56081	447	119
34	2005-05-27	58353	462	93
35	2005-06-27	57021	449	23
36	2005-08-01	57045	449	23
37	2005-08-30	58364	458	26
38	2005-09-28	58957	473	42
39	2005-10-31	59071	472	41
40	2005-11-29	59853	472	36
41	2005-12-20	62363	495	57
42	2006-01-31	65520	520	72

# Itération	Date	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (SLOC)	Nbre Classes	Fautes
43	2006-02-28	69218	563	65
44	2006-03-30	69690	574	71
45	2006-04-28	70600	582	60
46	2006-05-26	70594	582	45
47	2006-06-27	71693	602	54
48	2006-08-01	74393	617	37
49	2006-08-29	75261	629	61
50	2006-09-26	78868	672	34
51	2006-11-01	81093	688	31
52	2006-11-28	79548	670	47

Annexe 1b-2 : Caractéristiques des captures pour PDE.UI au niveau des classes

# Itération	Date	Valeurs moyennes des métriques OO pour les classes								
		Qi	LOC	CBO	RFC	LCOM	NOC	WMC	DIT	NOO
1	2002-08-29	0.774	78.7	2.61	15.5	9.7	1.00	8.5	0.88	4.17
2	2002-10-01	0.772	76.6	2.62	15.5	9.0	1.01	8.3	0.89	4.04
3	2002-10-29	0.771	77.4	2.64	15.5	9.3	0.99	8.4	0.88	4.11
4	2002-11-26	0.765	81.5	2.84	16.6	13.2	1.06	9.0	0.96	4.57
5	2002-12-17	0.767	81.8	2.91	16.5	12.6	1.08	8.7	0.97	4.46
6	2003-01-29	0.770	83.1	2.92	16.8	12.9	1.10	8.7	0.98	4.46
7	2003-02-27	0.768	83.7	2.94	17.0	13.3	1.13	8.8	0.99	4.48
8	2003-03-27	0.764	83.7	2.95	17.2	13.5	1.11	8.9	1.01	4.54
9	2003-04-29	0.762	83.7	2.96	17.1	13.4	1.09	9.0	0.99	4.55
10	2003-05-27	0.764	83.7	2.91	17.1	13.2	1.10	8.9	0.99	4.51
11	2003-07-01	0.762	84.2	2.88	16.5	13.0	1.12	8.9	0.97	4.53
12	2003-07-29	0.765	83.4	2.87	16.5	12.9	1.13	8.9	0.98	4.50
13	2003-08-28	0.798	83.3	2.83	16.6	13.0	1.09	8.9	0.99	4.53
14	2003-09-30	0.798	83.8	2.87	16.6	12.9	1.11	8.8	0.99	4.52
15	2003-10-29	0.801	80.6	3.09	16.7	15.3	1.02	8.7	0.90	4.55
16	2003-11-25	0.732	80.9	9.68	65.4	19.3	0.79	11.8	2.76	5.98
17	2003-12-18	0.678	125.4	15.69	79.4	41.4	0.98	17.0	2.67	7.78
18	2004-01-30	0.677	122.2	15.25	76.9	41.3	1.04	16.9	2.60	7.78

19	2004-02-26	0.702	117.4	14.93	76.7	40.3	1.00	16.4	2.67	7.66
20	2004-03-30	0.690	123.8	16.16	79.7	44.1	1.09	17.3	2.76	8.14
21	2004-04-28	0.681	128.0	17.17	84.5	46.0	1.14	18.0	2.94	8.35
22	2004-05-28	0.682	123.7	17.53	89.6	42.5	1.09	18.3	3.11	8.34
23	2004-06-24	0.678	124.8	17.62	90.2	43.7	1.11	18.5	3.12	8.40
24	2004-07-27	0.678	124.8	17.60	90.2	43.7	1.11	18.5	3.12	8.40
25	2004-08-19	0.675	125.3	17.73	89.7	43.3	1.08	18.8	3.09	8.43
26	2004-09-23	0.677	125.2	17.71	90.1	43.6	1.08	18.6	3.11	8.41
27	2004-11-01	0.674	126.1	17.82	90.0	44.0	1.08	18.9	3.09	8.45
28	2004-11-30	0.680	125.2	17.59	90.5	43.9	1.08	18.9	3.13	8.39
29	2004-12-21	0.676	125.3	17.30	90.8	44.7	1.09	19.2	3.14	8.49
30	2005-02-01	0.680	125.2	17.89	92.9	45.3	1.16	19.3	3.24	8.54
31	2005-03-01	0.674	128.4	18.36	93.8	47.8	1.15	20.0	3.26	8.76
32	2005-03-31	0.675	126.7	18.28	93.8	46.5	1.15	19.8	3.27	8.69
33	2005-04-26	0.673	125.5	19.12	92.7	46.2	1.20	19.8	3.19	8.81
34	2005-05-27	0.675	126.3	18.27	93.6	46.1	1.15	19.7	3.26	8.68
35	2005-06-27	0.670	127.0	19.49	93.6	46.8	1.21	20.1	3.22	8.86
36	2005-08-01	0.670	127.0	19.51	93.6	46.9	1.21	20.1	3.22	8.87
37	2005-08-30	0.670	127.4	19.54	93.9	47.1	1.22	20.2	3.23	8.90
38	2005-09-28	0.670	124.6	19.52	92.7	45.3	1.21	19.9	3.20	8.85
39	2005-10-31	0.682	125.2	19.76	92.1	42.5	1.16	19.5	3.25	8.50
40	2005-11-29	0.692	126.8	20.18	94.3	42.5	1.21	19.4	3.34	8.49
41	2005-12-20	0.714	126.0	20.16	93.9	41.9	1.23	19.3	3.34	8.41

42	2006-01-31	0.717	126.0	20.21	92.6	40.3	1.22	19.4	3.29	8.41
43	2006-02-28	0.722	122.9	24.17	99.7	49.7	1.15	22.5	3.24	9.17
44	2006-03-30	0.721	121.4	24.10	98.9	49.2	1.18	22.3	3.21	9.08
45	2006-04-28	0.722	121.3	23.92	98.5	49.5	1.19	22.5	3.21	9.05
46	2006-05-26	0.722	121.3	23.92	98.5	49.5	1.19	22.5	3.21	9.05
47	2006-06-27	0.721	119.1	23.53	95.9	49.8	1.24	22.3	3.15	8.96
48	2006-08-01	0.717	120.6	23.46	95.3	50.0	1.23	22.7	3.13	9.02
49	2006-08-29	0.718	119.7	23.23	95.6	50.2	1.24	22.6	3.14	9.00
50	2006-09-26	0.721	117.4	22.89	93.8	48.2	1.22	22.1	3.10	8.84
51	2006-11-01	0.722	117.9	22.92	94.1	48.5	1.23	22.3	3.10	8.91
52	2006-11-28	0.725	118.7	23.17	93.4	48.6	1.07	22.4	3.02	8.86

Annexe 1c-1 : Caractéristiques des releases pour Apache Tomcat (branche 5.5) au niveau du système

# Itération	Release	Valeurs moyennes des métriques OO pour les systèmes		
		Taille (KLOC)	Nbre Classes	Fautes
1	5.5.0	126927	837	6
2	5.5.1	127662	842	4
3	5.5.2	112784	842	1
4	5.5.3	129143	850	6
5	5.5.4	130893	857	22
6	5.5.5	132237	861	2
7	5.5.6	132339	861	6
8	5.5.7	135466	901	32
9	5.5.8	136257	906	5
10	5.5.9	138448	917	79
11	5.5.10	148470	961	9
12	5.5.11	148353	962	2
13	5.5.12	148895	968	41
14	5.5.13	149537	964	6
15	5.5.14	150550	979	25
16	5.5.15	148400	954	10
17	5.5.16	162383	1064	20
18	5.5.17	165043	1090	44
19	5.5.20	169621	1129	58
20	5.5.23	170886	1134	32
21	5.5.25	171627	1134	9
22	5.5.26	171916	1134	23
23	5.5.27	172285	1137	27
24	5.5.28	173102	1141	24
25	5.5.29	169622	1103	4
26	5.5.30	170025	1105	2
27	5.5.31	170178	1105	6
28	5.5.32	170215	1105	4
29	5.5.33	170163	1104	9
30	5.5.34	170854	1106	3
31	5.5.35	170998	1108	0

Annexe 1c-2 : Caractéristiques des releases pour Apache Tomcat (branche 5.5) au niveau des classes

# Itération	Release	Valeurs moyennes des métriques OO pour les classes								
		Qi	LOC	CBO	RFC	LCOM	NOC	WMC	DIT	NOO
1	5.5.0	0.743	151.6	8.97	60.5	178.7	0.686	29.8	1.60	12.7
2	5.5.1	0.745	151.6	8.96	60.4	179.1	0.690	29.7	1.60	12.7
3	5.5.2	0.745	133.9	8.17	56.3	68.2	0.815	25.6	1.60	10.6
4	5.5.3	0.753	151.9	8.91	60.6	172.9	0.692	29.7	1.60	12.5
5	5.5.4	0.752	152.7	8.95	60.5	173.1	0.690	29.9	1.60	12.6
6	5.5.5	0.750	153.6	9.02	61.1	173.1	0.688	30.2	1.60	12.6
7	5.5.6	0.750	153.7	9.03	60.8	173.1	0.688	30.2	1.60	12.6
8	5.5.7	0.753	150.4	9.03	61.0	168.4	0.696	29.4	1.63	12.5
9	5.5.8	0.753	150.4	9.04	61.5	168.3	0.696	29.4	1.63	12.5
10	5.5.9	0.751	151.0	9.04	61.3	167.7	0.697	29.5	1.63	12.5
11	5.5.10	0.746	154.5	9.21	63.3	180.6	0.701	30.3	1.65	13.0
12	5.5.11	0.744	154.2	9.21	63.6	177.5	0.702	30.2	1.65	12.9
13	5.5.12	0.744	153.8	9.19	63.8	176.7	0.706	30.2	1.66	12.9
14	5.5.13	0.745	155.1	8.49	58.3	183.6	0.713	30.4	1.45	13.1
15	5.5.14	0.743	153.8	9.16	63.7	177.8	0.703	30.2	1.66	13.0
16	5.5.15	0.742	155.6	9.23	63.8	181.2	0.717	30.5	1.66	13.0
17	5.5.16	0.743	152.6	9.29	64.1	177.2	0.686	30.0	1.66	13.0
18	5.5.17	0.743	151.4	9.31	65.6	175.5	0.685	29.8	1.67	12.9

19	5.5.20	0.744	150.2	9.27	65.4	172.2	0.671	29.5	1.67	12.7
20	5.5.23	0.738	150.7	9.25	65.7	174.4	0.676	29.7	1.68	12.8
21	5.5.25	0.737	151.3	9.26	65.9	177.4	0.677	29.8	1.68	12.9
22	5.5.26	0.737	151.6	9.26	66.1	177.7	0.677	29.9	1.68	12.9
23	5.5.27	0.737	151.5	9.25	66.1	177.3	0.676	29.8	1.68	12.8
24	5.5.28	0.737	151.7	9.28	66.1	177.5	0.674	29.8	1.68	12.8
25	5.5.29	0.735	153.8	9.38	65.9	180.2	0.666	30.3	1.64	13.0
26	5.5.30	0.735	153.9	9.37	65.8	181.0	0.665	30.3	1.64	12.9
27	5.5.31	0.735	154.0	9.37	65.9	183.4	0.665	30.3	1.64	13.0
28	5.5.32	0.735	154.0	9.37	65.9	183.4	0.665	30.3	1.64	13.0
29	5.5.33	0.735	154.1	9.37	66.0	183.6	0.664	30.3	1.64	13.0
30	5.5.34	0.734	154.5	9.39	66.1	183.8	0.663	30.3	1.64	13.0
31	5.5.35	0.735	154.3	9.41	65.9	183.7	0.661	30.3	1.64	13.0