Impact des bonnes pratiques sur la qualité du code

Salma Hamza

▶ **To cite this version:**

Salma Hamza. Impact des bonnes pratiques sur la qualité du code. Génie logiciel [cs.SE]. 2011. dumas-00636417

HAL Id: dumas-00636417
https://dumas.ccsd.cnrs.fr/dumas-00636417

Submitted on 27 Oct 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Impact des bonnes pratiques sur la qualité du code

HAMZA Salma

Encadrant : Salah Sadou
Laboratoire : VALORIA

VALORIA

Equipe : SE

Encadrant : Stéphane Oberlechner
Entreprise : Agrostar

AGROSTAR

Département : Méthodes

Master recherche en informatique
Architecture et ingénierie logicielles des systèmes autonomes et adaptatifs

Juin 2011

Résumé

Il existe des bonnes pratiques applicables à tous les niveaux du développement, allant de l'analyse jusqu'au codage. Mais qu'en est-il de leur impact réel sur la qualité du code produit. Dans ce rapport nous allons mener une étude dont l'objectif est de déterminer la variation de la qualité du code selon les bonnes pratiques utilisées. Cette étude, nous a permis de définir une bonne pratique de modélisation, adaptée à un contexte industriel particulier, afin d'améliorer la qualité des logiciels produits.

Abstract

There are so many good practices to each level of developement ; from analysis to coding. But ; what is their real impact on code quality. In This study, our aim is to determine the variation of code quality according to good practices used. This study will enable us to define a good modelling practice in a particular industrial context ; in order to improve produced software quality.

Remerciements

Avant d'entamer la présentation de ce travail, je voudrais d'abord adresser mes sincères remerciements et reconnaissances à tous ceux qui m'ont si efficacement aidé à passer au mieux mon stage.
En particulier, à mon encadreur à VALORIA Equipe SE Salah SADOU, de qui j'aurais encore beaucoup à apprendre, souhaitant qu'il trouve ici l'expression de ma profonde reconnaissance pour sa disponibilité, ses critiques, ses précieux conseils qu'il m'a prodigués tout au long de ce travail.
Je tiens à remercier également mon encadreur à Agrostar Stéphane OBERLECHNER, souhaitant qu'il trouve ici l'expression de ma gratitude pour sa patience, son assistance et suivi incessant par ses directives et ses conseils précieux.
Mes remerciements vont également à toute l'équipe SE, en particulier Vincent LEGLOAHEC et Kahina HASSAM aussi bien pour leurs aides et leurs explications que pour leurs qualités humaines et leurs suggestions.

Je tiens à remercier toute l'équipe d'Agrostar pour m'avoir accueilli pendant ces 5 mois et plus particulièrement Corentin MENOU pour le temps qu'il a su me consacrer et les conseils avisés qu'il a su me donner.
Je remercie mes amis Henda et Babacar pour leurs soutiens.
Je remercie ma famille et plus particulièrement ma sœur et mon frère.

Je termine par un grand remerciement à mes parents auxquels je dédie ce travail.

Table des matières

Remerciements 1

Introduction générale 6

1 Etat de l'art 8

1.1 Qu'est ce qu'un logiciel de qualité ? 8

1.2 La crise du logiciel 9

1.3 Facteurs de qualité 9

1.4 Métriques 10

1.4.1 Quelques exemples de métriques 11

1.4.2 Utilité des métriques 11

1.4.3 Normalisation des métriques 11

1.4.4 Outils d'évaluation de la qualité 13

1.5 Démarches d'amélioration de la qualité 14

1.5.1 Bonnes pratiques 15

2 Modélisation des bonnes pratiques 18

2.1 GooMod : Langage de modélisation des BPs 18

2.1.1 Syntaxe abstraite de GooMod 19

2.1.2 Architecture générale de la plateforme GooMod 19

2.2 Définition de BP 21

2.2.1 Les contraintes 22

2.2.2 Les actions 23

2.2.3 Les concepts 23
2.3 Exemple de définition de BPs 23

3 Etude de cas **27**
3.1 Processus du développement d'un logiciel sans BP 27
3.2 Processus du développement d'un logiciel avec BP 28
3.3 Interprétation des résultats 29

4 Etude emirique **33**
4.1 Métriques 33
4.1.1 Métriques de classe 33
4.1.2 Métriques de méthode 34
4.2 Evaluation des métriques 36
4.2.1 Etude de la corrélation 36
4.2.2 Analyse des résultats 36

Conclusion **39**

Bibliographie **40**

Annexes **44**

A Tableaux **44**

Liste des figures

1.1 Facteurs de qualité ISO 9126. 10
1.2 Modèle de normalisation 12
1.3 Modèle d'évaluation de la qualité 13
1.4 Mesure préventive et amélioration de la qualité du code 14
1.5 Architecture d'évaluation du système de qualité. 17

2.1 Syntaxe abstraite du langage GooMod. 19
2.2 Architecture générale de la plateforme GooMod. 20
2.3 Processus définie via GooMod. 21
2.4 Exemple d'un message de sortie. 24

3.1 Processus basé sur la mesure de la qualité. 27
3.2 Processus basé sur BP. 28
3.3 Facteurs de qualité mesurés par CAST. 29
3.4 Comparaison des mesures. 29
3.5 Les critères du facteur de qualité Transferabilité. 30
3.6 Métrique du convention du nommage des packages (2 versions du code). 30
3.7 Ensemble de critères du facteur de qualité Changeabilité. 31
3.8 Ensemble de critères du facteur de qualité Sécurité. 31
3.9 Ensemble de critères du facteur de qualité Robustesse. 32

4.1 Métriques de classe. 34
4.2 Métriques de méthode. 35
4.3 Corrélations possibles entre les métriques. 37

A.1 Corrélation entre les métriques de classe. 44
A.2 Corrélation entre les métriques de méthode. 45

Introduction générale

De nos jours les systèmes informatiques sont de plus en plus complexes. Face à cette complexité, les spécialistes s'intéressent de plus en plus aux conséquences de l'évolution des logicielles, activité incontournable, sur leur qualité. En effet, la qualité a toujours suscité l'intérêt aussi bien les chercheurs, dans le domaine du domaine du génie logiciel, que des développeurs dont le souci est d'assurer les exigences de leurs clients.

Beaucoup d'études sont menées pour trouver des méthodes efficaces pour l'amélioration de la qualité du logiciel : plusieurs métriques **[Chidamber et al., 1994]**, **[Brito et al., 1994]**, plusieurs modèles de qualité **[AliKacem et Sahraoui, 2006]** et plusieurs normes **[Masud al., 2008]**,**[ISO , 2001]** ont été proposés afin de contribuer à la production de logiciels de qualité. L'une des méthodes qui peut nous aider à améliorer la qualité du produit logiciel est l'application des bonnes pratiques.

L'étude des bonnes pratiques nécessite la collecte des données dans un contexte industriel. Il se trouve qu'un partenaire industriel (Agrostar) a décidé de se doter d'un processus assurant le développement de logiciel de qualité. Dans ce projet de recherche, il s'agit d'utiliser cette opportunité pour définir un processus d'identifier l'impact des bonnes pratiques de modélisation sur la qualité du code produit. L'intérêt pour notre partenaire industriel est d'avoir la définition d'une bonne pratique de modélisation lui assurant un certain niveau de qualité de son code.

Le présent rapport est divisé en cinq chapitres : Le premier chapitre est consacré à l'état de l'art. Dans une première partie, de ce chapitre, nous présenterons quelques notions fondamentales sur la qualité des logiciels, ensuite nous évoquerons quelques exemples de métriques, dans le double but d'illustrer la problématique de la mesure et de montrer son influence sur le code. La deuxième partie vise à donner un aperçu sur les bonnes pratiques appliquées aux systèmes logiciels. Dans le deuxième chapitre nous présenterons notre approche pour la définition d'une bonne pratique de modélisation adaptée au contexte industriel définit par Agrostar. Dans ce

même chapitre, nous présenterons le langage GooMod, un langage permettant la définition et l'application des bonnes pratiques de modélisation. Le troisième chapitre, sera consacré à l'exploitation des résultats obtenus. Dans le dernier chapitre, nous ferons une analyse empirique des métriques pour vérifier et valider notre contribution.

CHAPITRE 1 Etat de l'art

Introduction

Dans le présent chapitre, nous présentons quelques notions fondamentales nécessaires pour mener une étude sur les bonnes pratiques et critique de l'existant. Nous commencerons par définir un certain nombre de notions lié à la qualité du logiciel.
Ensuite, nous dévoilerons le besoin de mesurer. Ainsi, nous présenterons l'état de l'art sur les différentes manières de spécifier et d'implémenter des métriques, Enfin, nous présentons un volet sur quelques outils de bonnes pratiques.

1.1 Qu'est ce qu'un logiciel de qualité ?

Il n'y a pas un accord universel sur le sens de la qualité d'un logiciel. En effet, il existe plusieurs définitions. En voici deux exemples :

Définition 1 : Selon le standard ISO 9000 **[ISO, 2000]**, la qualité d'un logiciel est l'ensemble des caractéristiques intrinsèques qui lui confère l'aptitude à satisfaire les besoins et attentes exprimés ou implicites des clients et autres parties intéressées.

Définition 2 : : Selon le standard IEEE **[IEEE, 1998]**, la qualité logicielle correspond au degré selon lequel un logiciel possède une combinaison d'attributs désirés.

Donc, la qualité ne réside pas uniquement dans le fait d'avoir un logiciel opérationnel, mais également dans l'évaluation par rapport à des objectifs mesurables de qualité.
En effet, Pour construire des produits de qualité, il nous faut des activités standardisées. A cet effet, plusieurs mesures de qualité et normes ont été proposées. Selon le standard ISO 9126, la qualité d'un produit est représentée par un ensemble de caractéristiques opérationnelles, qu'on appelle facteurs de qualité.

1.2 La crise du logiciel

Les problèmes du développement du logiciel ont été identifiés par la crise du logiciel (*software crisis*). Ses principaux symptômes sont :

- Le logiciel n'est pas délivré à temps et les coûts dépassent le budget alloué, le dépassement de délai et de coût moyen est compris entre 50 et 70 %.
- La qualité du logiciel correspond rarement aux attentes des clients.
- Les modifications des logiciels coûtent cher et sont à l'origine de nouveaux défauts.
- 50 % des projets n'aboutissent jamais.

Ces dernières indications montrent à quel point ce secteur d'ingénierie a besoin de s'améliorer d'où plusieurs travaux de recherche ont été réalisés pour pallier à cette crise.

1.3 Facteurs de qualité

Pour qu'un logiciel soit de bonne qualité, il doit répondre à un nombre important de critères qui sont représentés dans la figure 1.1.

La qualité du logiciel est définie par six facteurs principaux décrits dans la norme ISO 9126.

- **Capacité fonctionnelle :** en quoi le logiciel satisfait les besoins exprimés ?
- **Fiabilité** : le produit fait-il tout le temps les tâches prévues avec la précision requise ?
- **Ergonomie :** est-il facile d'utiliser le logiciel ?
- **Efficacité :** quels sont les performances du logiciel ?
- **Maintenabilité :** Est-il facile d'identifier et corriger les composants en dysfonctionnement ou des données erronées ?
- **Portabilité :** est-il facile d'adapter le logiciel et le transférer dans un autre environnement matériel ou sur une nouvelle configuration ?

Chaque facteur est relié à des caractéristiques qui ont un groupement de métriques.

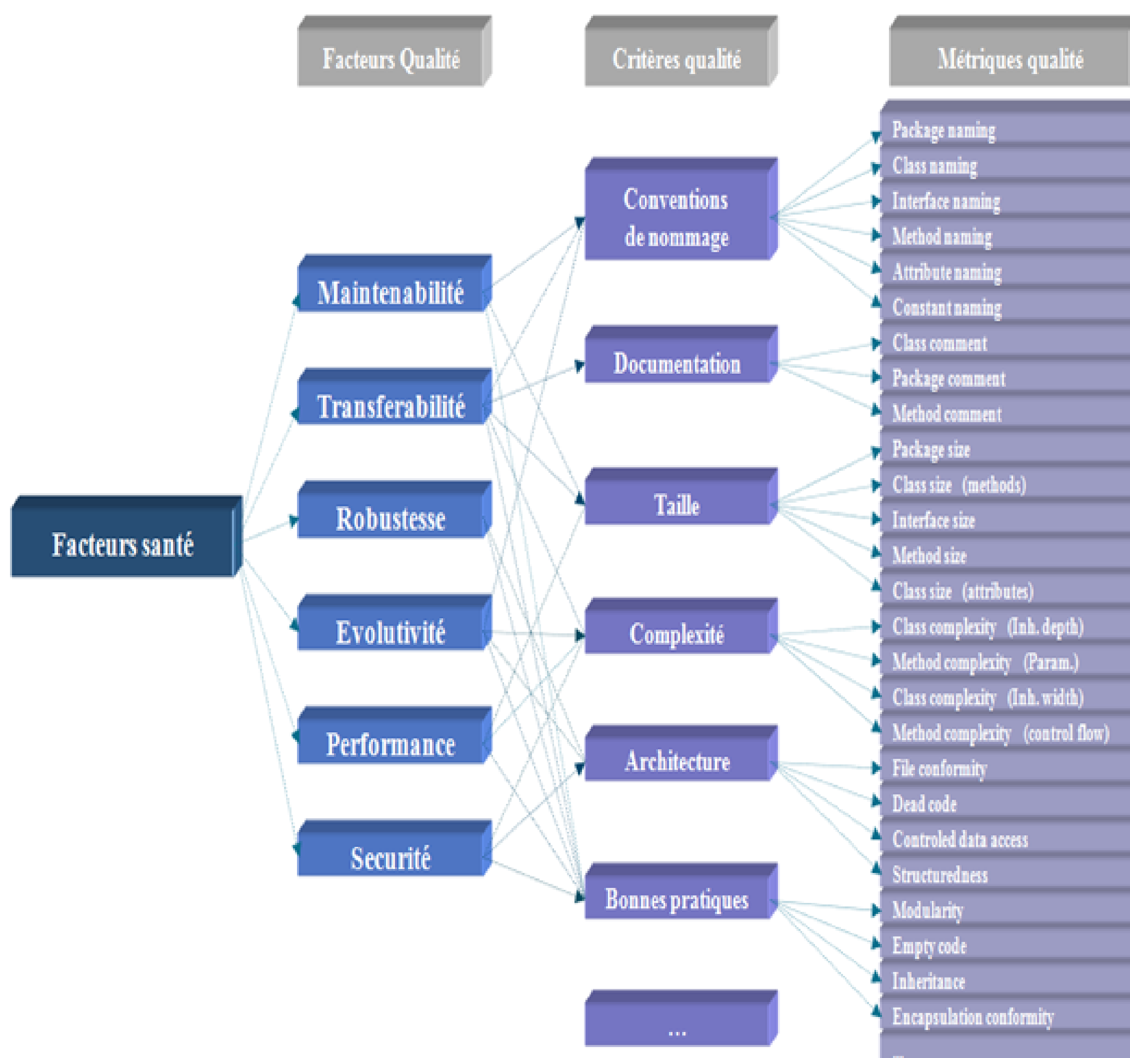


Figure 1.1 — Facteurs de qualité ISO 9126.

1.4 Métriques

Les performances des processus logiciels sont un facteur important à mesurer. L'évaluation de ses performances s'appuie sur des métriques. Ces métriques ont été intégrées dans la pratique. Ceci est dû au fait que généralement les chercheurs préfèrent prendre leurs décisions en s'appuyant sur des données mesurables " *Vous ne pouvez pas gérer ce que vous ne contrôlez pas, et vous ne pouvez pas contrôler ce que vous ne mesurez pas* " **[DeMarco, 1982]**.

Une grande liste de métrique a été mise à notre disposition pour évaluer la structure et la qualité d'un système logiciel. Cela à créer la nécessité de déterminer quel ensemble de mesure est le plus approprié pour chaque contexte.

1.4.1 Quelques exemples de métriques

Il existe de nombreuses métriques bien connues qui mesurent par exemple la taille du code, la structure des produits logiciels ("Complexité cyclomatique" ou "Fan-in/Fan-out"). Chidamber et al. **[Chidamber et al. 1994]** définissent six métriques pour les classes dans le cas des logiciels orientés-objets sont les plus cités en génie logiciel **[Wohlin, 2007]** :

- WMC : la somme des complexités de toutes les méthodes.
- DIT : la profondeur de l'héritage de la classe.
- NOC : le nombre de classes immédiatement dérivées d'une classe.
- CBO : la mesure de couplage correspondant au nombre de classes dont la classe considérée est dépendante.
- RFC : le nombre de méthodes qui peuvent être potentiellement exécutées en réponse à un message.
- LCOM : la mesure du manque de cohésion, c'est le nombre de paires de méthodes qui n'utilisent aucune variable d'instance en commun.

Basili et al. **[Basili et al. ,1996]** ont montré empiriquement que ces métriques sont de bons indicateurs pour prédire la probabilité de faute.
Brito et Carapuça **[Brito et al., 1994]** ont proposé une autre type de classification, ils ont classifié les métriques selon six dimensions (conception, taille, complexité, réutilisabilité, productivité et qualité) et ont proposé trois niveaux de granularité (méthode, classe et système). Il en résulte 18 classes de métriques.

1.4.2 Utilité des métriques

Une étape indispensable à l'amélioration du domaine de la mesure est de déterminer l'utilité de la mesure pour chaque logiciel. En effet, on ne mesure pas un logiciel pour le plaisir de mesurer mais pour des objectifs bien définis.
Oman et Pfleeger **[Oman et al. ,1997]** ont distingué six objectifs pour mesurer. Ces objectifs sont : mesurer pour la compréhension, mesurer pour 1'expérimentation, mesurer pour le contrôle de projet, mesurer pour l'amélioration du processus, mesurer pour l'amélioration du produit et mesurer pour la prédiction.

1.4.3 Normalisation des métriques

Les métriques de produits logiciels ont été intégrées dans la pratique, vu leurs importances. Bansiya et Davis **[Bansiya et al. , 1997]** ont développé des métriques originales pour leurs projet QMOOD (*Quality Model for Object-Oriented Design*). Toutefois, comme il n'y a aucune

norme, il est difficile de déterminer les valeurs des seuils métriques.
L'approche de la normalisation **[Masud et al. , 2008]** fournit une plate-forme standard pour les métriques de la qualité des systèmes logiciels sans tenir compte de la qualité de leur conception, de la complexité et de la taille. Cette approche, comme le montre la figure 1.2, se base sur trois niveaux fondamentaux.

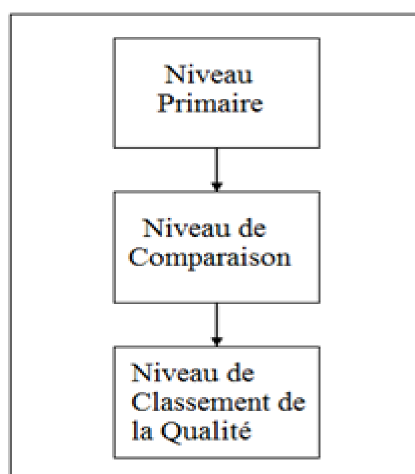


***Figure 1.2* —** Modèle de normalisation .

La normalisation au niveau primaire : représente la valeur de métrique calculée en nombre de lignes de code indifféremment de leur taille.

La normalisation au niveau de comparaison : en cas de mesure comparative de la qualité entre les systèmes logiciels avec des métriques similaires, la normalisation de certaines valeurs de métriques est recommandée.

La normalisation au niveau de classement de la qualité : s'appuyant sur les deux techniques de normalisation du niveau primaire et du niveau de comparaison. Ce type de normalisation est effectué pour attribué le classement de la qualité de conception pour tout système logiciel.

L'objectif de la normalisation est de fournir les informations générales sur les indicateurs de qualité des logiciels, de créer un standard sur la mesure de qualité logicielle et d'automatiser le procédé. Toutefois, les résultats ont été ambigus. " Nous concluons que les modèles existants sont incapables de prédire avec précision les défauts en utilisant que des métriques. En outre, ces modèles de métriques n'expliquent pas comment corriger les défauts" **[Fenton et al., 1999]**

1.4.4 Outils d'évaluation de la qualité

Les outils de métriques ont pour objectif d'identifier et de compter les erreurs du code source et par conséquent, de donner un résultat synthétique sur la qualité du code [Wagner et al. , 2008]. La norme ISO 9126 [ISO, 2001] définit les outils d'évaluation comme étant un ensemble d'attributs de qualité liés à un ensemble de métriques. La relation entre les attributs de qualité et les métriques précise le processus d'évaluation de qualité.
La figure 1.3 présente le modèle d'évaluation de la qualité du logiciel. Il existe trois niveaux d'évaluation :

- MÉTRIQUES : Indicateurs de propriétés élémentaires comparés à un seuil.
- CRITÈRES : Indicateurs de caractéristiques du produit Ic = f (m1, m2, m3, ...)
- FACTEURS : Indicateurs d'objectifs à atteindre If = g (Ic1, Ic2, Ic3, ...) Les valeurs seuils associé à chaque mesure ne doivent pas être dépassées. Ces valeurs sont comparées aux résultats des mesures pour évaluer la qualité.

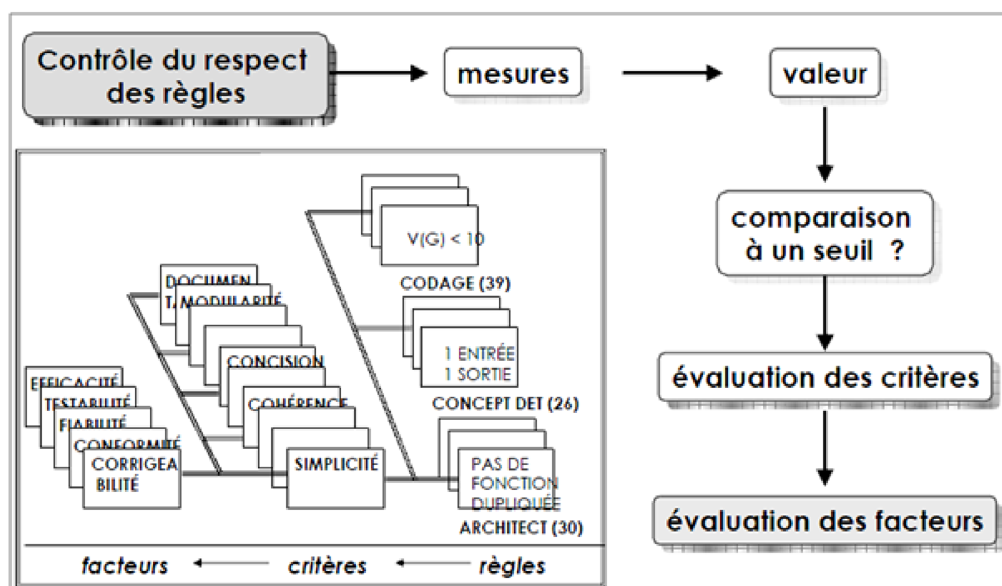


Figure 1.3 — Modèle d'évaluation de la qualité .

Les outils de l'évaluation qualité permettent de comparer l'état du produit et du processus de développement par rapport aux dispositions initiales(les seuils). Ces outils sont les plus proches de la contribution de ce travail. Notons que les valeurs seuils seront les valeurs du standard ISO 9126.
Dans cette partie, nous avons présenté un état de l'art sur les métriques pour des systèmes du logiciel. Il convient de souligner qu'il n'existe pas un outil de métriques universel pour aboutir à un logiciel de bonne qualité. De plus, la qualité du logiciel n'est pas absolue, mais est estimée en fonction des besoins de chaque projet. Dans la prochaine partie nous examinerons les bonnes

pratiques qui pourront nous aider à améliorer la qualité du logiciel.

1.5 Démarches d'amélioration de la qualité

De nombreuses approches ont déjà été proposées sur la qualité des processus des systèmes d'information. La plupart affirment qu'il y a un fort besoin pour des mesures appropriées de logiciel dès les premières phases du développement des logiciels, tels que l'analyse des besoins, spécification, conception, et la partie initiale du codage. De ce fait, le cycle de vie d'un logiciel et l'assurance qualité sont fortement lié.
Il a été démontré aussi **[Boehm, 1981]** que les erreurs de conception sont extrêmement coûteuses lorsqu'elles ne sont pas corrigées dans les premières phases du cycle de vie des projets. Dès lors, il devient donc important de pouvoir contrôler la qualité d'un logiciel au début de la phase de conception.

Le but principal de la mesure en phase amont du développement logiciel est la prédiction de la qualité finale du logiciel pour réduire les coûts et éviter les erreurs de conception. Les activités d'assurance de la qualité, pour ce type, sont l'inspection plutôt que le test [Tian , 1998]. Ce type de mesure permet de valider l'architecture du logiciel, sachant que les propriétés de celle-ci influent sur les caractéristiques finales de qualité (voir figure 1.4).

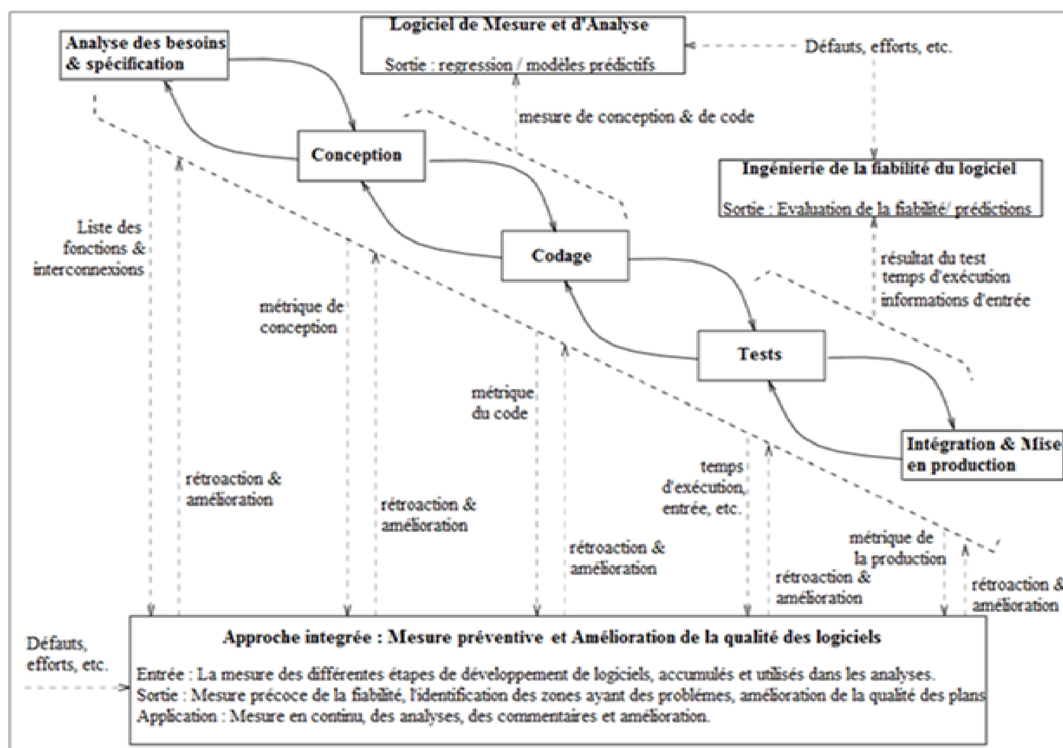


Figure 1.4 — Mesure préventive et amélioration de la qualité du code .

1.5.1 Bonnes pratiques

Pour produire des logiciels de qualité une méthodologie des bonnes pratiques a été mise en place. [LeGloahec et al. , 2009] Les bonnes pratiques sont des techniques et des méthodes, rassemblées par une entreprise ou par une communauté, dont l'efficacité a été constatée lors des projets de développement.
Les bonnes pratiques présentent l'expertise et la valeur ajoutée de l'entreprise, c'est une assurance de la conformité et du respect de la norme.
L'ingénierie dirigée par les modèles, permet de rendre les bonnes pratiques productives et permet la vérification automatique de leurs respects.

Lignes directrices

D'après les recherches antérieures qui ont été faites pour l'évaluation de la qualité du logiciel, les chercheurs ont montré que la taille du modèle d'un processus, sa structure, l'expertise de son concepteur ainsi que d'autres facteurs ont un impact sur le modèle du processus. A cet effet, une série de lignes directrices a été présenté **[Mendling et al., 2010]**.
Les spécialistes du domaine ont défini une relation entre les caractéristiques structurelles d'un modèle de processus et les différents facteurs de compréhension y compris la mesure de compréhension d'un modèle de processus, la probabilité d'erreur et de l'ambiguité. Donc, évaluer un processus revient à déterminer certaines mesures (cohérence, lisibilité...).
Les sept lignes directrices de modélisation (*Seven Process Modeling Guidelines*) (7 PMG) fournissent un ensemble de recommandations sur la façon de construire un modèle de processus pour l'améliorer. Chacune des lignes directrices s'appuie sur des recherches empiriques décrites ci-dessous. Il est important de noter que les 7 PMG s'appuient sur l'idée qu'il y a différentes façons pour décrire le même comportement pour un même modèle de processus.

Les 7 PMG sont les suivantes **[Mendling et al., 2010]** :

- 1 : Utiliser le moins d'éléments dans le modèle de processus que possible.
- 2 : Minimiser les chemins de routage de chaque élément.
- 3 : Utiliser un unique événement de départ et un unique événement de fin.
- 4 : Structurer le modèle le mieux possible.
- 5 : Éviter les éléments de routage OR.
- 6 : Utiliser des étiquettes d'activité verbe-objet.
- 7 : Décomposer le modèle s'il a plus de 50 éléments.

Les 7 PMG sont utiles pour guider les utilisateurs à améliorer la qualité de leurs modèles et pour devenir plus compréhensible. Chacune de ces lignes donne des directives sur la façon dont un modèle de processus peut être amélioré. Donc, les 7 PMG permettent d'améliorer l'efficacité

des projets au sein des entreprises qui s'appuient sur ce type de modèles conceptuels pour modéliser leurs systèmes.
Il est important de noter que l'application des 7 PMG ne touche pas la logique qui se trouve derrière le modèle original.

Langages dédiés aux bonnes pratiques

Un langage de spécification du domaine (DSL) est conçu pour répondre à une problématique spécifique, contrairement au langage de modélisation UML qui vise une généralité de problématiques.
Dans la littérature, les langages de spécification des bonnes pratiques telles que les processus de modélisation **[Perez et al., 2007]**, les mesures, les styles **[Ambler , 2010]** et les modèles peuvent fournir des outils essentiels et complémentaires à la fois.
Le langage de spécification des bonnes pratiques impose une façon d'exécuter une activité ou un processus, et impose aussi un certain nombre de contraintes sur l'utilisation appropriée des concepts. Ce type de langage est initialement destiné à la modélisation, mais il est tout à fait adaptable à d'autres phases du développement, à condition de fournir le méta-modèle de l'activité visée.

Plusieurs langages ont été proposés dans la littérature pour définir les règles à suivre pour la conception et la mise en œuvre d'un programme orienté objet de qualité.
L'approche d'Alikacem et Sahraoui [Alikacem et Sahraoui, 2006] consiste à détecter les anomalies en utilisant un langage de règle de qualité. Ils définissent un langage de description de métriques (orienté c++ mais supporte aussi java) et un méta-modèle de logiciel orienté objet.

La figure 1.5 représente l'architecture proposée par Alikacem et Sahraoui. Cette architecture est composée de trois constituants assurant : (1) la représentation du code source, (2) la collecte des métriques et (3) la détection de non conformités.
Marinscu et al. **[Marinscu et al., 2005]** spécifient une nouvelle approche pour l'implémentation des métriques : un DSL dédié aux métriques orientées objet. Ce type de DSL, automatise la mesure des programmes de grandes tailles.
Beyer et al. **[Beyer et al., 2005]** ont proposé un nouveau langage nommé RML (Relation Manipulation Language) dont le but de détecter des patrons de conception, des défauts de design, et de calculer des métriques.
Un langage récent proposé par l'équipe SE du Valoria **[LeGloahec et al., 2010]**, nommé GooMod, fournit les propriétés nécessaires à la description des meilleures pratiques de conception, ce langage nous permet de s'assurer du respect des bonnes pratiques lors de la production d'un code.

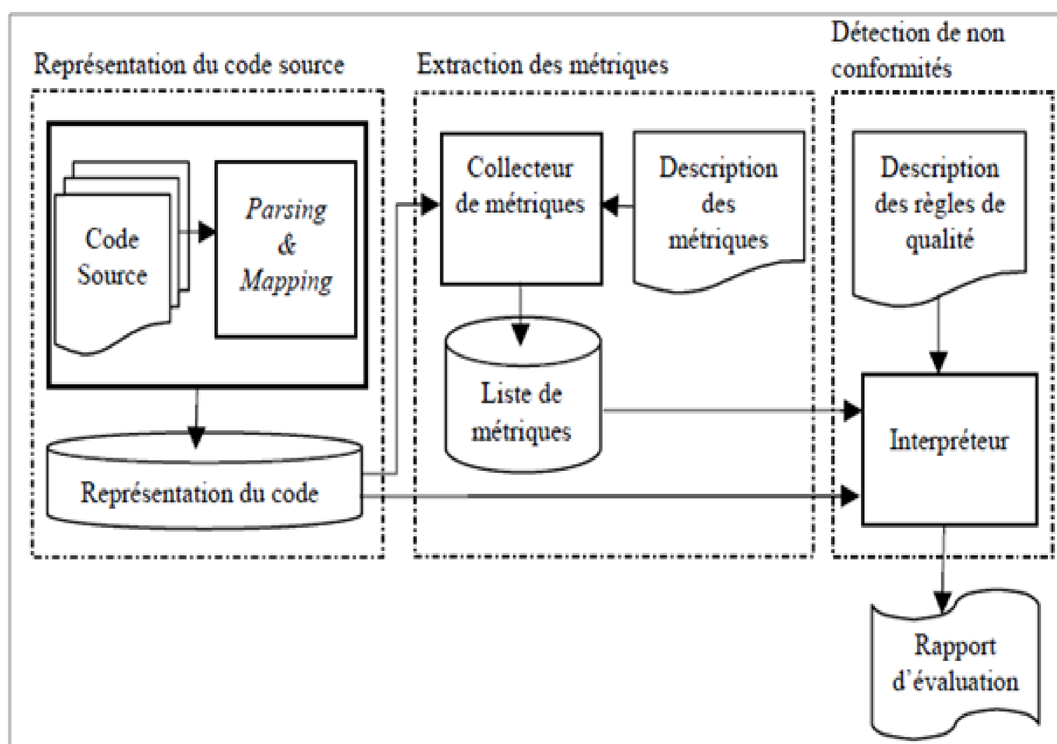


Figure 1.5 — Architecture d'évaluation du système de qualité.

Conclusion

Dans cette partie, nous avons présenté un état de l'art sur les métriques du logiciel. Il convient de souligner qu'il n'existe pas un outil de métriques universel pour aboutir à un logiciel de bonne qualité. De plus, nous avons présenté les bonnes pratiques et les nombreuses études empiriques qui ont été introduites. Dans les chapitres suivant nous déterminons leurs impacts réels sur la qualité du code produit.

CHAPITRE 2

Modélisation des bonnes pratiques

Introduction

Afin de faire face à la complexité des logiciels, de nombreuses approches ont déjà été proposées sur la qualité des processus des systèmes logiciels. Parmi ces approches il y a celles qui s'occupent des bonnes pratiques de la modélisation. Ces bonnes pratiques sont des techniques et des méthodes, rassemblées par une entreprise dont l'efficacité a été constatée lors des projets de développement.
Dans ce chapitre, nous introduisons le langage de bonne pratiques GooMod, ensuite, nous présentons sa syntaxe abstraite ainsi que l'architecture de la plateforme qui le supporte. Enfin, nous définissons une bonne pratique de modélisation, adaptée à un contexte industriel particulier qui est Agrostar.

2.1 GooMod : Langage de modélisation des BPs

GooMod (*Good Modeling Practices*) **[LeGloahec et al., 2010a]** est un langage de spécification des bonnes pratiques de modélisation (développé en utilisant la plateforme Eclipse), issu des travaux de l'équipe SE du Valoria. Ce langage est associé à tout type de DSL (*Domain Specific Languages*) dont la syntaxe abstraite est décrite dans un modèle MOF (*Meta Object Facility*). L'outil supportant ce langage a été conçu comme un assistant de modélisation pour aider les concepteurs à créer des modèles de qualité. De plus il est indépendant des outils de modélisation.

2.1.1 Syntaxe abstraite de GooMod

Le méta-modèle du langage GooMod est décrit dans la figure 2.1.

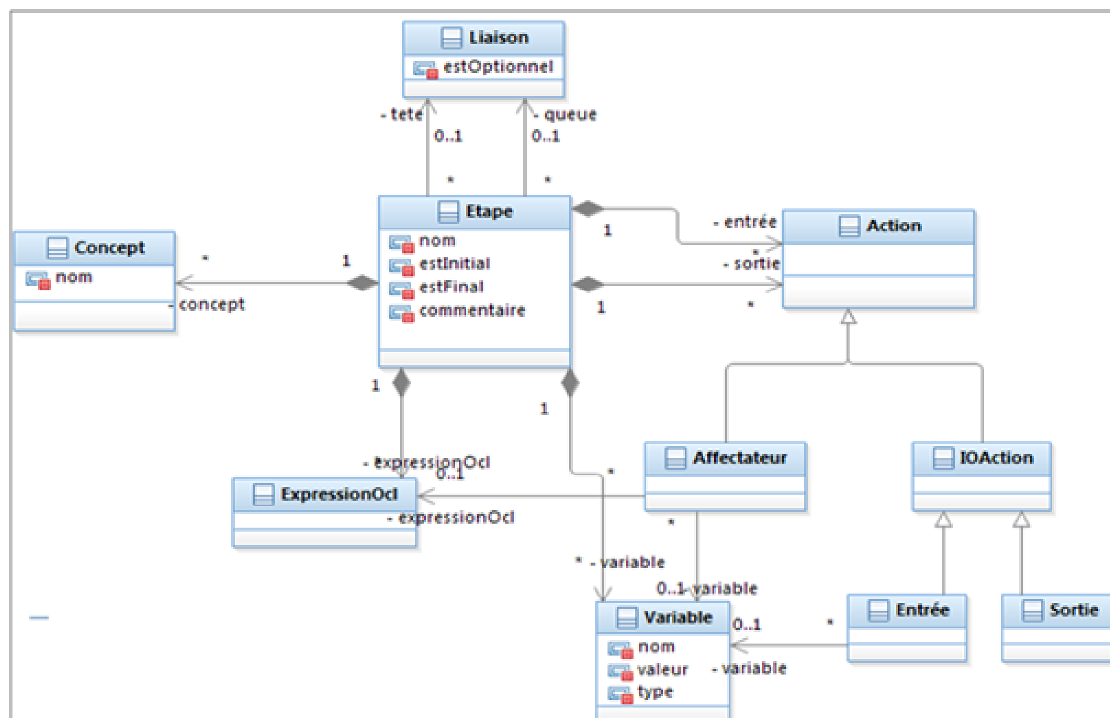


Figure 2.1 — Syntaxe abstraite du langage GooMod.

Une étape est associée à quatre éléments : son contexte, le type de modélisation associé, l'ensemble des concepts qui seront utilisables lors de son exécution et un ensemble d'actions. GooMod permet à son utilisateur d'intégrer des actions au début ou à la fin de l'étape. En effet, à chaque étape il peut être nécessaire de disposer des données supplémentaires que seul le concepteur peut prévoir. De même, le système est capable de nous fournir des informations qu'ils ne sont pas visible sur le modèle et qu'on ne peut pas en déduire facilement. De plus, les actions permettent une interaction avec le concepteur à l'aide des messages.
Les arcs peuvent être marqués comme facultatifs, ce qui signifie que l'étape est facultative.

2.1.2 Architecture générale de la plateforme GooMod

Le langage GooMod est supporté par une plateforme complète pour la gestion des BP de la modélisation, allant de leurs définitions au niveau PIM jusqu'à leurs adoptions au niveau PSM. La figure 2.2 illustre la plate-forme et sa séparation PIM-PSM. Cette section décrit les deux niveaux et leurs outils associés.

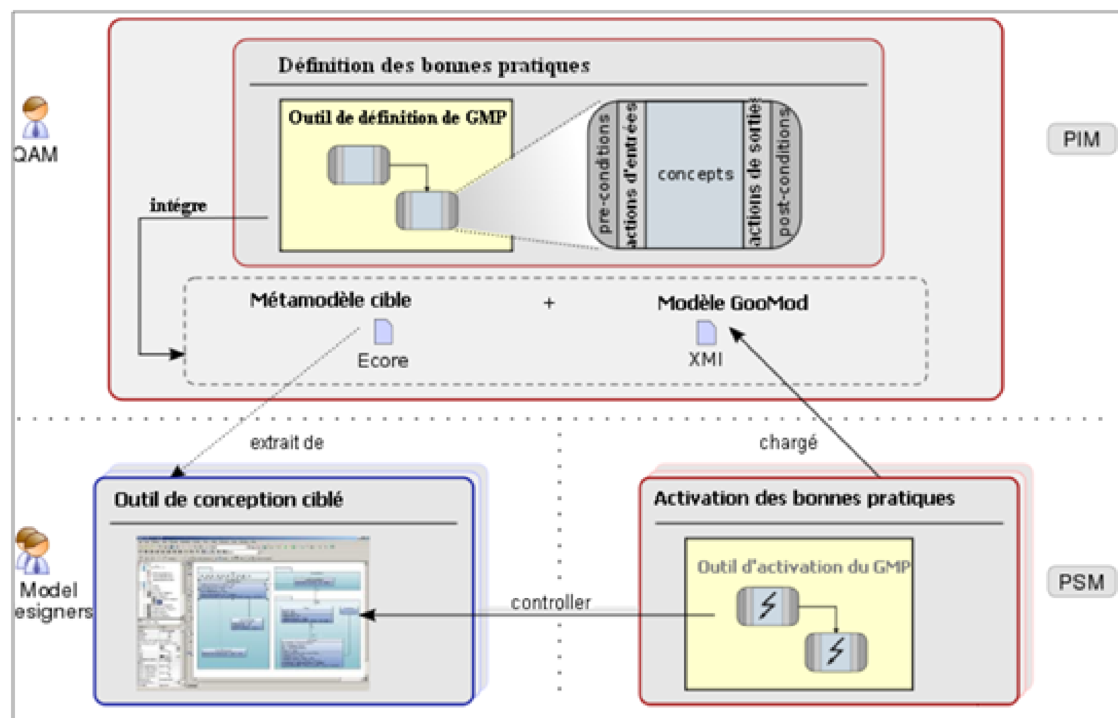


Figure 2.2 — Architecture générale de la plateforme GooMod.

Niveau PIM

Le niveau PIM (Platform Independent Model) désigne uniquement la partie modèle de l'application. C'est un modèle spécifiant une application indépendamment de toute plate-forme technique.
Le niveau PIM du langage GooMod permet la description des BP indépendamment de l'outil de conception utilisée, il ne contient pas d'information sur les technologies utilisées pour déployer l'application.
Cet outil est conçu pour QAM (Quality Assurance Managers) chargé de la définition de BP (voir figure 2.2).

Eclipse Modeling graphic Framework (GMF), permet la représentation du BP sous la forme d'un processus. Un tel processus est représenté sous forme de graphe.
Une fois le concepteur a décrit son modèle de BP, le modèle est stocké dans un format XMI(XML Metadata Interchange) afin que d'autres outils puissent l'utiliser.

Niveau PSM

Le niveau PSM (Plateform Specific Model) sert essentiellement à la génération de code exécutable vers la plate-forme technique. Il décrit le mode d'emploi de cette plate-forme.

Dans le cas du GooMod, le niveau PSM vise à fixer la définition des bonnes pratiques déduit dés le niveau PIM. Pour cela, notre plate-forme est composée de deux parties :

– **Outil d'activation des BPs :** vise à associer le modèle de BP définie en utilisant l'outil de définition de GMP (Good Modeling Process), avec l'outil de modélisation approprié.
– **Outil de conception ciblé :** c'est l'outil de conception où les BP seront effectuées. Cet outil ne peut pas être modifié directement, il sera contrôlé par un plugin externe, qui est dans notre cas l'outil d'activation GMP.

A travers cette partie, nous avons vu que GooMod est un langage assez puissant pour définir les bonnes pratiques. Il est adaptable pour n'importe quel éditeur de base. Dans la section suivante, nous décrivons un exemple de BP.

2.2 Définition de BP

Le langage GooMod permet de représenter les différentes étapes de BP sous la forme d'un processus. Ainsi, nous avons décrit un processus de développement de logiciel avec le langage GooMod, dont le but d'assurer la production et/ou l'amélioration d'un logiciel de qualité. Les étapes sont décrites dans la figure 2.3.
Le QAM chargé de la définition des BP est en mesure de détailler chaque étape du processus en y ajoutant quelques règles qui doivent être vérifiées.

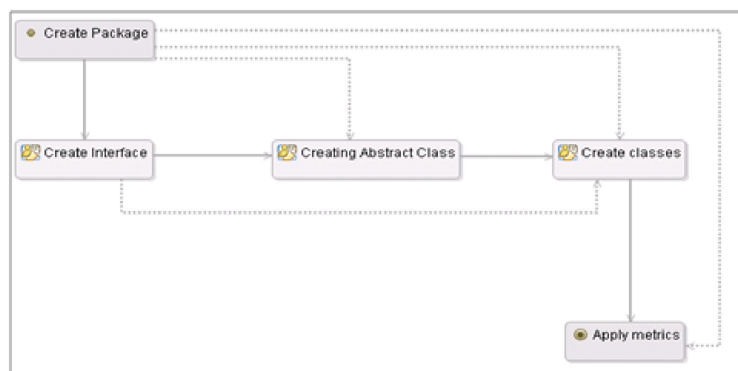


Figure 2.3 — Processus définie via GooMod.

Dans notre processus nous définissons des règles de BP, organisées sous la forme de graphe voir figure 2.3. Ils sont précisément définit en se basant sur le méta-modèle UML 2.0.
En réalité, nous avons utilisé un diagramme UML réduit aux concepts réellement utilisés par le contexte de l'entreprise Agrostar. Ainsi, le modèle utilisé est un DSL.

2.2.1 Les contraintes

Afin de garantir une bonne construction du modèle, nous pouvons définir des pré- et/ ou post-conditions pour chaque étape de BP. L'application des conventions de codage est une étape de base dans le processus d'amélioration de la qualité et l'efficacité. Le langage GooMod est utilisé pour établir les conventions de codage dés la phase de la modélisation. Ces conventions ont été définit par le langage OCL(*Object Constraint Language*). Ce langage de spécification formelle, permet de définir syntaxiquement et sémantiquement des restrictions sur les modèles exprimés dans des notations graphiques, étendant ainsi la capacité expressive de telles notations. De cette façon, les schémas complétés par des expressions OCL sont plus exacts et complets.

Contraintes qualitatives

L'avantage avec GooMod que l'amélioration de la lisibilité du code du logiciel est forcée, ce qui permet après aux membres de l'équipe de comprendre le code plus rapidement et en profondeur. Pour une bonne lisibilité du code, les conventions de nommage doivent être respectées. Exemple de convention du nommage pour l'interface :

```
Interface.allInstances()->forAll(i:Interface|i.name.substring(1,1)='I')
```

En dépit de la limite du langage OCL, nous avons pu mettre en place une contrainte qui peut nous renseigné sur quelques complexité dû au polymorphisme et l'héritage pour l'amélioration de la qualité.

```
Class.allInstances()->forAll(c:Class|c.interfaceRealization->size()<=4)
```

Contraintes quantitatives

Avec GooMod, la vérification de la conformité de la valeur mesurée avec le standard ISO 9126 est assurée, par exemple un package ne doit pas avoir plus que 30 éléments, cette contrainte est classée parmi les métriques du standard ISO 9126 comme contrainte lié à la complexité due au polymorphisme et l'héritage.

```
Package.allInstances()->forAll(p:Package|p.ownedElement->
select(e:Element|e.oclIsTypeOf(Class)and e.oclIsTypeOf(Interface)) ->size()<=30)
```

2.2.2 Les actions

Une liste des actions d'entrées et de sorties est associée à chaque étape du modèle des BP. Les actions d'entrées sont lancées juste après la vérification de la pré-condition. Elles permettent l'initialisation de l'environnement associé à cette étape. De même, lorsque le concepteur indique qu'il a finit de cette étape, une liste d'actions de sortie sera lancé pour préparer l'environnement de l'étape à cette fin. Par conséquent, les actions présentent un moyen d'interaction entre le concepteur et le modèle des BP.

2.2.3 Les concepts

Dans une étape donnée, le concepteur est limité par un ensemble de concept. Chaque concept lui a été associé différents types d'utilisation (création, suppression, mise à jour).

2.3 Exemple de définition de BPs

Pour définir la BP adapter au besoin du développement chez Agrostar, nous avons procédé comme suit :

- Analyser l'ensemble de métriques utilisées pour mesurer la qualité du code produit.
- Déduire les propriétés des métriques analysées.
- Identifier la méthode de développement pour la décomposer en étape. La méthode du développement est proche de l'OMT(*Object Modeling Technique*).
- A chaque étape identifier l'ensemble de propriétés concernées par les métriques. Le processus de BP se compose ainsi de 5 étapes, nous avons associé un contexte à chacune de ses étapes, un ensemble d'actions et de concepts. Pour assurer la concision et la qualité du modèle, de nombreuses pré et post-conditions ont été définis :

1 ére étape :

Créer des packages : au cours de cette étape initiale, seul le concept package du méta-modèle UML sera disponible.
Une contrainte de convention du nommage doit être préservé : Les noms des packages doivent être en minuscule, selon la norme ISO 9126. Une action de sortie souligne les éléments qui ne respectent pas cette contrainte :

```
Package.allInstances() -> select (p :Package|p.name<> p.name.substring(1,p.name.size()).toLower()).name
```

La contrainte ci-dessus permet d'identifier toutes les packages du modèle, qui ne respectent pas la convention. La figure 2.4 montre la forme du message destiné au concepteur.

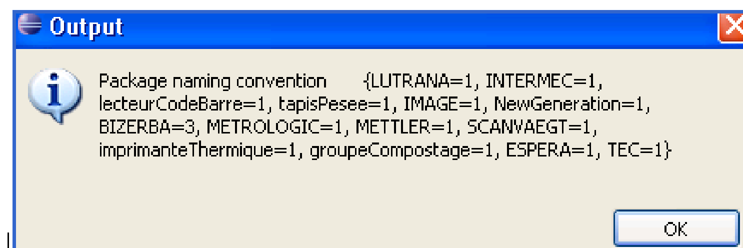


Figure 2.4 — Exemple d'un message de sortie.

2 éme étape :

Créer des interfaces : à cette étape, seuls les concepts d'interface, d'opération, et de liaisons, propres à cette étape, sont mises à disposition du concepteur. Ainsi, Avant d'aborder cette étape, il faut s'assurer qu'au moins un package a été ajouté au modèle. Pour ce faire, une pré-condition est définie pour cette étape :

```
Package.allInstances ()-> notEmpty ()
```

De même pour l'interface, une Post-condition a été définit pour vérifier la contrainte de convention de nommage : Les noms des interfaces doivent obligatoirement commencer par la lettre 'I' :

```
Interface.allInstances()->forAll(i:Interface|i.name.substring(1,1)='I')
```

Une autre Post-condition a été décrite pour cette étape et qui concerne un aspect quantitatif issu aussi de la norme ISO 9126. Il s'agit de vérifier que le nombre des opérations de l'interface ne dépassent pas vingt.

```
Interface.allInstances()->forAll(i:Interface|i.ownedOperation->size()<=20)
```

3 éme étape :

Créer des classes abstraites : Dans les pratiques de développement chez Agrostar, chaque matériel est associé à une classe abstraite et par suite, dans cette étape, le concepteur devrait envisager de créer des classes abstraites. Les concepts d'UML retenus pour cette étape sont les classes et la propriété d'abstraction, les opérations et les attributs. La convention de nommage doit être préservée pour les classes abstraites. Le nom de la classe abstraite doit commencer par

la lettre 'A' pour les différencier des classes d'implémentation. Cette règle a été définit comme une post-condition :

```
Class.allInstances()->forAll(c:Class| c.isAbstract implies c.name.substring(1,1) ='A')
```

D'autres règles, se basant aussi sur la norme ISO 9126 concernant le nombre des opérations et des attributs des classes abstraites qui ne doivent pas dépasser les trente :

```
Class.allInstances()->select(c:Class|c.ownedOperation->size()<=30)
```

```
Class.allInstances()->forAll(c:Class|c.ownedAttribute->size()<=30)
```

4 éme étape :

Créer des classes à cette étape, les concepts de classes, d'attribut, d'opération, et de liaisons, propres à cette étape, sont mises à disposition du concepteur. Ainsi, nous avons définit quelques contraintes quantitatives pour assurer la qualité selon le standard 9126 , il faut s'assurer que la classe contient au moins un attribut et au plus trente.

```
Class.allInstances()->forAll(c:Class|c.ownedAttribute->size()->notEmpty())
```

```
Class.allInstances()->forAll(c:Class|c.ownedAttribute->size()<=30)
```

Il faut vérifier de même, qu'au moins il y a une opération dans la classe et au plus trente :

```
Class.allInstances()->forAll(c:Class|c.ownedOperation->size()->notEmpty())
```

```
Class.allInstances()->forAll(c:Class|c.ownedOperation->size()<=30)
```

La deuxième règle est de mentionner les classes qui implémentent trop d'interfaces.

```
Class.allInstances()->forAll(c:Class|c.interfaceRealization->size()<=4)
```

5 ème étape :

Appliquer des métriques : nous avons consacré la dernière étape à la vérification de certaines aspects que nous ne pouvons pas vérifier au cours des autres étapes. Par exemple, pour vérifier la taille d'un package (la taille ne doit pas dépasser les trente éléments en totalité). Il est évident que cette règle ne peut être vérifiée qu'une fois tout le processus est terminé.

```
Package.allInstances()->forAll(p:Package|p.ownedElement->
select(e:Element|e.oclIsTypeOf(Class)and e.oclIsTypeOf(Interface)) ->size()<=30)
```

Conclusion

Définir les BP n'est pas une tâche aisée. Dans ce chapitre, nous nous sommes appuyés sur le standard ISO 9126 et l'ensemble de métrique utilisé par Agrostar pour la définition d'une telle BP. Dans le chapitre suivant, nous allons voir l'impact de ses BP sur la qualité du logiciel.

CHAPITRE 3 Etude de cas

Introduction

Afin d'améliorer la qualité de ses logiciels produits, Agrostar, filiale du groupe STEF-TFE, a besoin d'améliorer le processus de développement logiciel.
Dans ce chapitre, nous présenterons le processus du développement d'un logiciel, chez Agrostar avec et sans la BP définie dans le chapitre précédent. Ensuite, nous nous intéresserons aux résultats des mesures de la qualité prise avec l'outil CAST dans les deux cas. Enfin, nous interpréterons les courbes obtenues de différents facteurs de qualité.

3.1 Processus du développement d'un logiciel sans BP

La mise en œuvre du programme de mesure à Agrostar a été motivée par des raisons de la qualité. Le but était d'améliorer la qualité du code en premier lieu. Nous présentons dans la figure 3.1 le processus du développement d'un logiciel dans Agrostar avant d'intégrer l'outil de BP.

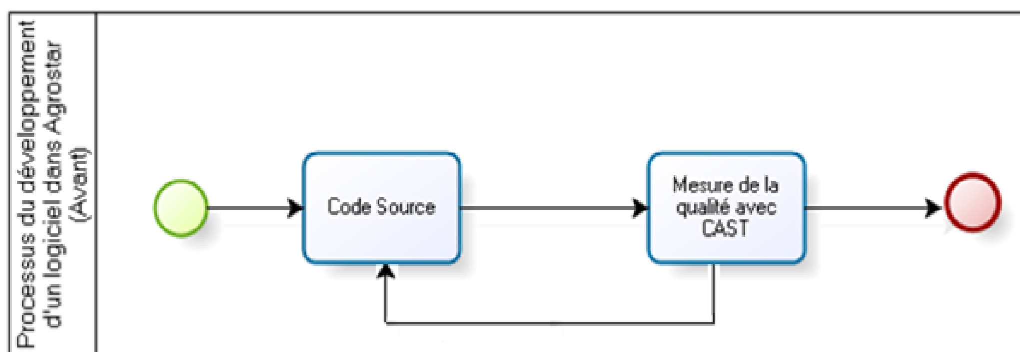


Figure 3.1 — Processus basé sur la mesure de la qualité.

Le processus du développement d'un logiciel dans Agrostar a été basé sur l'outil de mesure

CAST. Le développeur évalue la qualité de son code via CAST. S'il obtient un mauvais résultat il revient à l'étape de codage pour améliorer la qualité. Donc ce processus basé sur un outil de mesure, reste incapable d'améliorer vraiment la qualité. Par conséquent, ce processus ne satisfait pas les responsables de qualité d'Agrostar.

3.2 Processus du développement d'un logiciel avec BP

Notre objectif est d'adapter le nouvel outil de BP basé sur le langage de modélisation GooMod au besoin d'Agrostar afin d'avoir une meilleure qualité. La figure 3.2 illustre le processus du développement d'un logiciel avec GooMod.

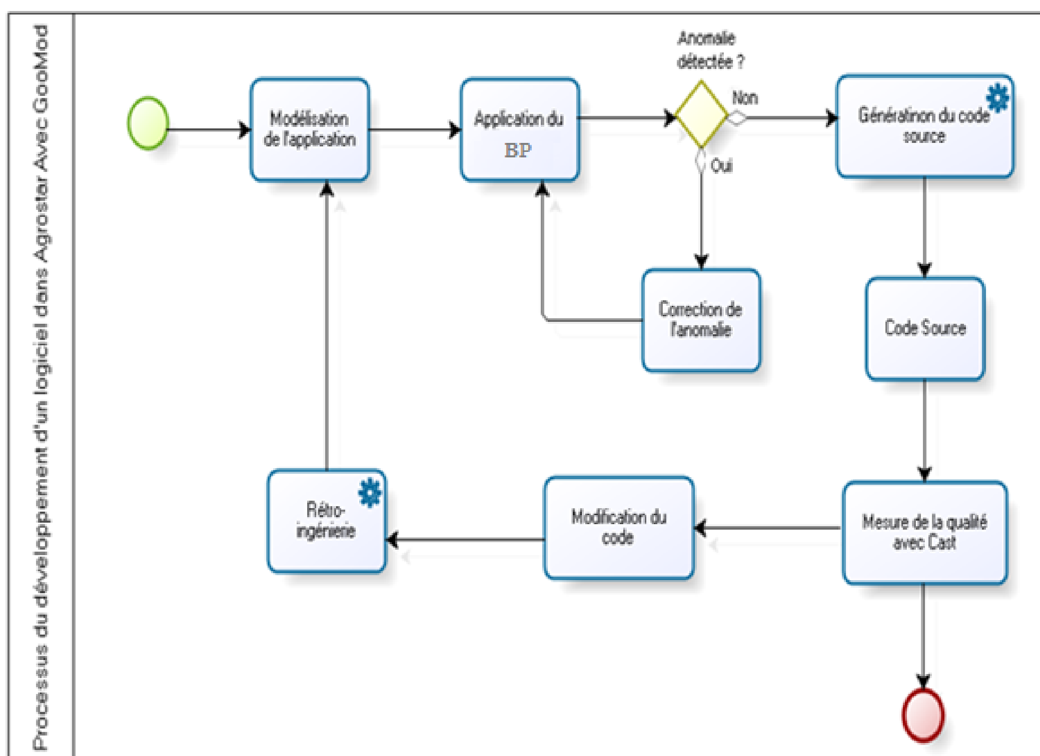


Figure 3.2 — Processus basé sur BP.

Avant de commencer à coder, le développeur doit impérativement modéliser son application, pour pouvoir appliquer l'outil de BP. Cet outil mentionne les défauts conceptuels qui régressent la qualité. Le développeur corrige les erreurs. Une fois le code est vérifié, le développeur génère son code source via un outil approprié. A la fin, le développeur évalue la qualité de son code généré avec CAST.

3.3 Interprétation des résultats

La figure 3.3 montre une capture d'écran des facteurs de qualités mesurées par l'outil CAST.

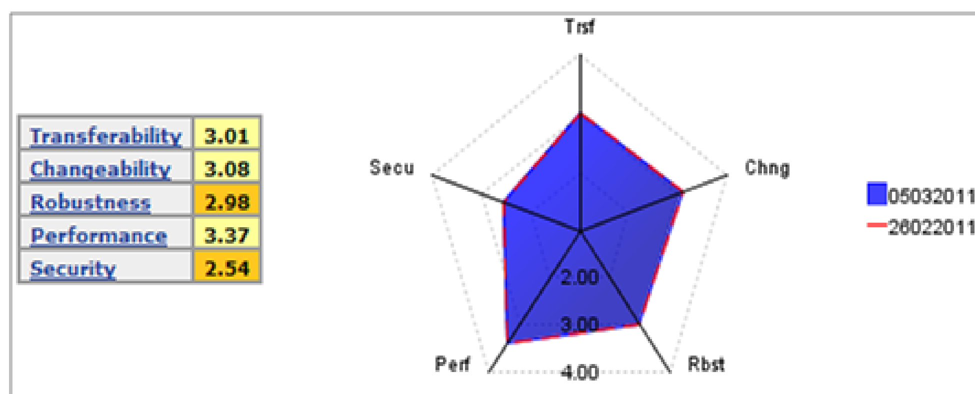


Figure 3.3 — Facteurs de qualité mesurés par CAST.

L'outil CAST permet de visualiser l'évolution de la qualité d'une version à une autre, ce qui nous permet d'évaluer si les objectifs ont été atteints et si la qualité du code a été améliorée. En comparant les deux versions, nous remarquons que la plupart des facteurs ont été améliorées.

Health Factor	Transferability	Changeability	Robustness	Performance	Security	Technical Quality Index
Poste prepa - Java Project - 02032011	2.89	2.99	3.08	2.86	2.71	2.92
Poste prepa - Java Project - 05052011	2.9	3	3.06	2.86	2.72	2.92

Figure 3.4 — Comparaison des mesures.

Nous nous basons dans la prochaine partie sur les facteurs de qualité, qui ont changé. Les facteurs de qualité ne peuvent pas être mesurés directement sur les produits logiciels. Ils sont mesurés en se basant sur un ensemble de critère. Les courbes ci-dessous montrent l'effet de BP sur la qualité du code.

Transferabilité : Parmi les critères du facteur transferabilité, figure le critère de conformité des conventions du nommage 'Documentation_Naming convention conformity' (voir annexe).Ce critère est amélioré de 0.09 (voir figure 3.5).

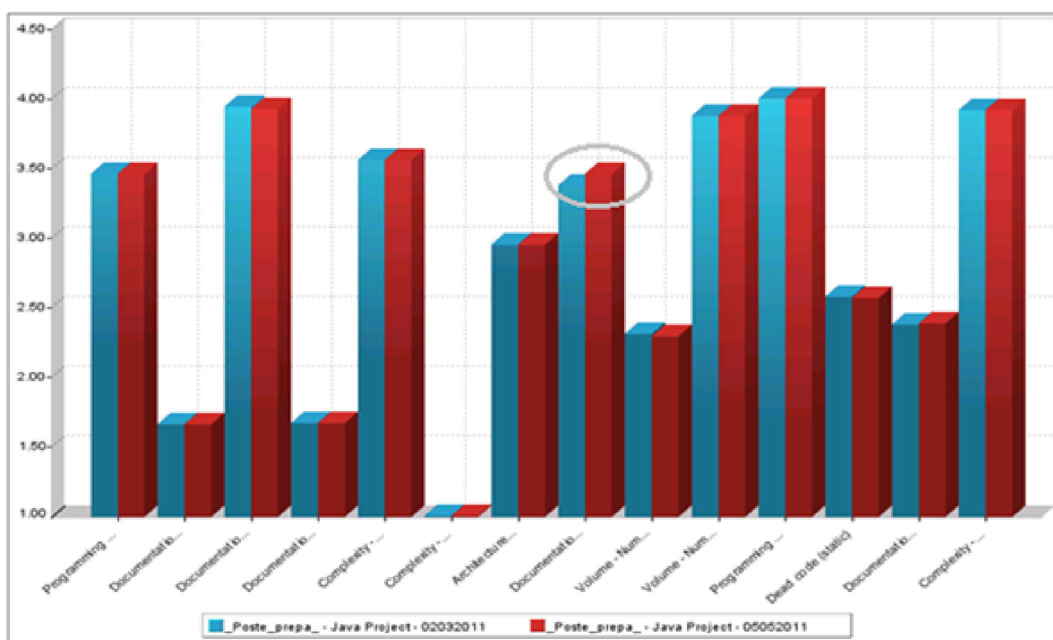
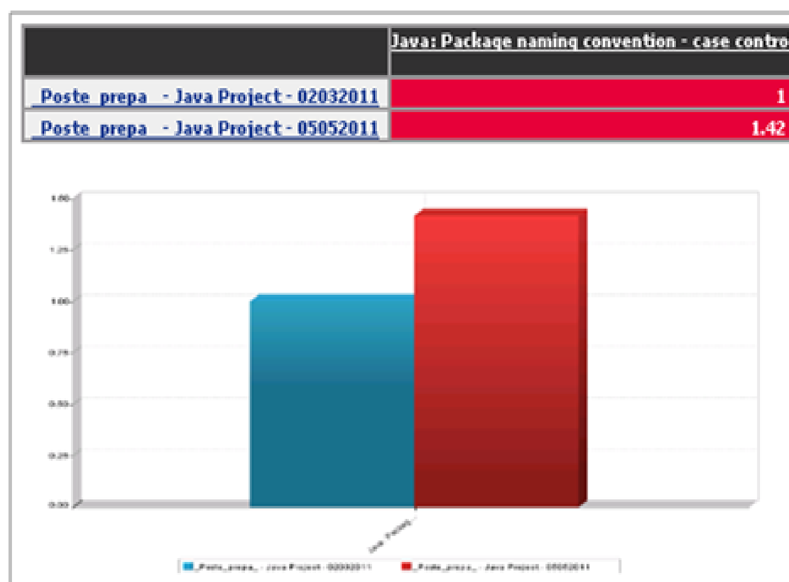


Figure 3.5 — Les critères du facteur de qualité Transferabilité.

Ce critère s'est amélioré grâceà l'amélioration de la métrique convention du nommage des packages " *Package naming convention*" . Dans la figure 3.6, nous pouvons déduire l'amélioration que nous avons pu la faire (0.42 de hausse) :



	Java: Package naming convention - case control
_Poste_prepa_ - Java Project - 02032011	1
_Poste_prepa_ - Java Project - 05052011	1.42

Figure 3.6 — Métrique du convention du nommage des packages (2 versions du code).

Une amélioration de métrique entraîne l'amélioration du critère et par conséquent, l'amélioration du facteur de qualité.

Changeabilité :

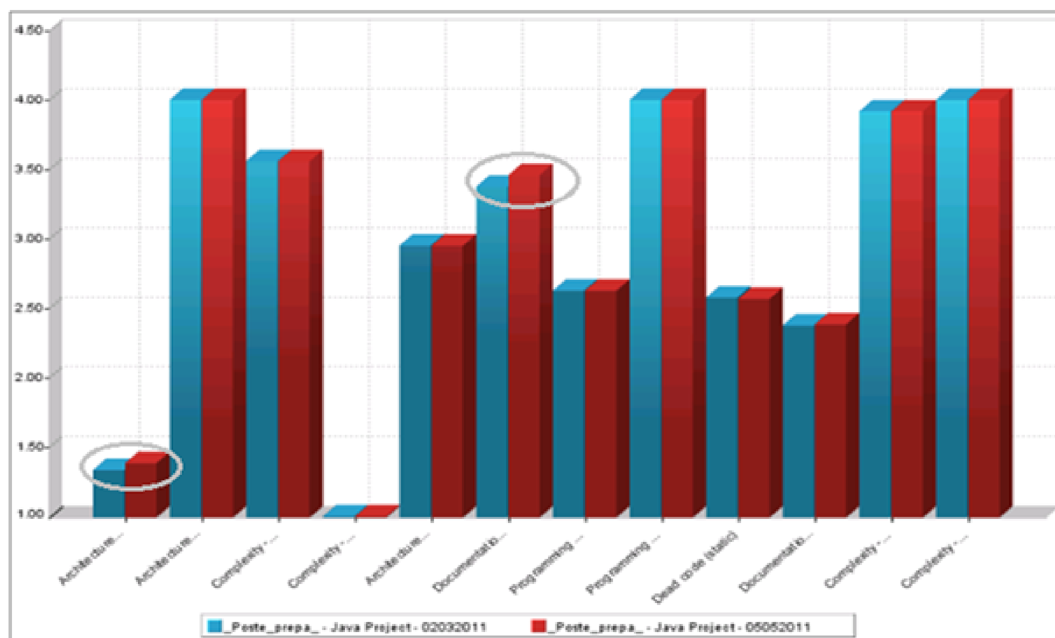


Figure 3.7 — Ensemble de critères du facteur de qualité Changeabilité.

Un accroissement a été enregistré également au niveau du facteur changeabilité (voir figure 3.7) du fait de l'amélioration des deux critères Architecture Models Automated Checks (0.05 de hausse) et ça est due au règle lié à la complexité qui a été mis en place et Naming Convention Conformity(0.09 de hausse).

Sécurité :

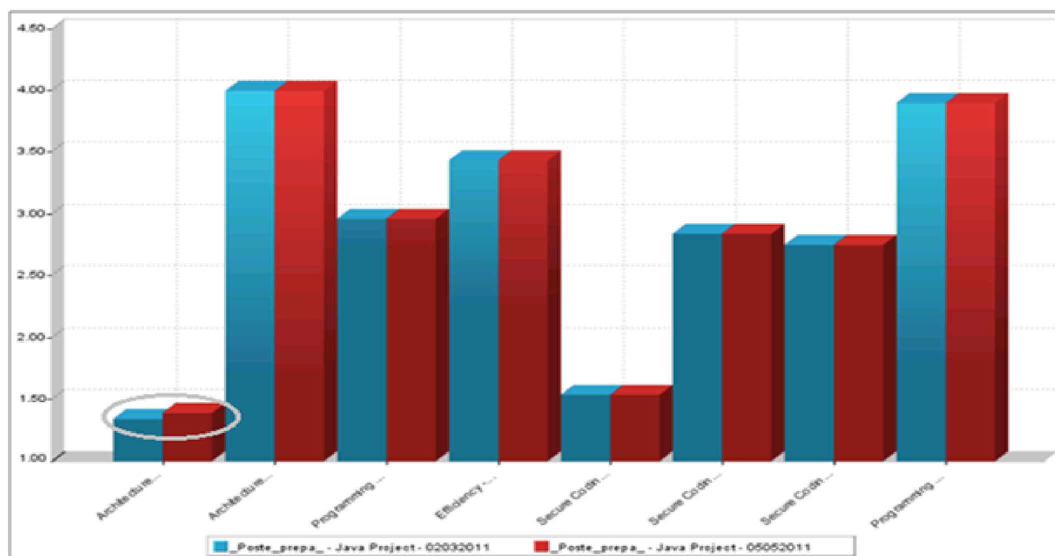


Figure 3.8 — Ensemble de critères du facteur de qualité Sécurité.

Notons que les critères Architecture existe dans la plupart des facteurs, le fait de l'améliorer et de jouer sur ces critères améliore nettement les résultats.

Robustesse :

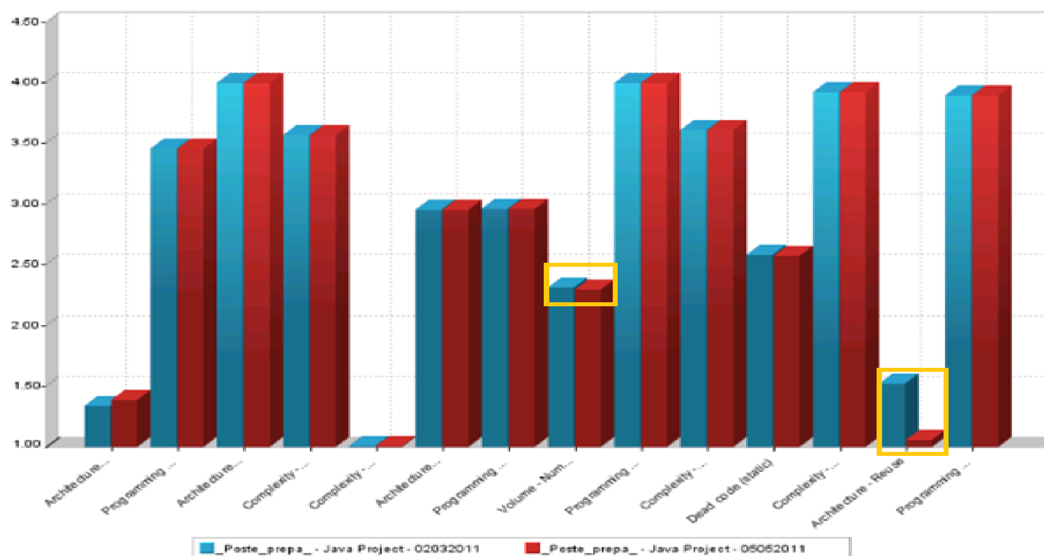


Figure 3.9 **—** Ensemble de critères du facteur de qualité Robustesse.

Par contre, le facteur de qualité Robustesse (figure 3.9) accuse une régression de (-0.02). Le recul observé dans la robustesse s'explique par le déclin enregistré dans le critère Réutilisabilité (Reuse) et le critère Volume_Nombre des composants (Volume_Number of Components). Le declin du 2éme critère s'explique par le fait que notre BP indique que la classe dépasse le nombre d'opération, nous éclatons dans ce cas la classe en 2 classes pour suivre les bonnes pratiques. Par contre, on n'est pas capable d'expliquer la baisse du critère Architecture (Reuse). La qualité du code a sensiblement évolué. En effet, nous avons enregistré une petite progression au niveau de trois facteurs de qualité. Cette progression pourrait être plus nette si on modifie tous les packages (nous avons appliqué le processus de BP sur une partie du code tandis que les mesures sont relatives à tous le code). Par contre on a vu que l'un des facteurs : Robustesse a baissé.

Conclusion

Les résultats obtenus sont encourageants même si on a eu un facteur de qualité qui a baissé. Pourquoi le facteur de robustesse a baissé ? Néanmoins, Pour mieux comprendre les résultats obtenus et trouver une explication à cette question, une étude empirique est toutefois nécessaire.

CHAPITRE 4 Etude emirique

Introduction

Les facteurs de qualité permettent de juger si le logiciel est de bonne ou de mauvaise qualité. Ils ne peuvent pas être mesurés directement sur les produits logiciels. Ils sont mesurés en se basant sur un ensemble de critère. Chaque critère est composé d'un ensemble de métriques. Notre étude, dans ce chapitre, portera sur les métriques et les impacts directs qu'elles peuvent avoir sur les facteurs de qualité.

4.1 Métriques

Il existe plusieurs types de métriques de qualité du code donc il est intéressant de les comparer. Afin d'effectuer des comparaisons de métriques et de juger leurs corrélations, il est indispensable de disposer de bases de données. La base dont on s'est appuyé est le code de la post-Préparation d'Agrostar.
Les principales études sont menées sur les métriques de classe et de méthode. L'évaluation est effectuée en utilisant 21 métriques de classe et 16 métriques de méthode.

4.1.1 Métriques de classe

Les métriques de classe nous renseignent sur la mesure de la qualité de la classe Java.

CE **Efferent Coupling**	Le nombre de classes dans un packages qui dépendent d'une classe d'un autre package.
CA **Afferent Coupling**	- Indique le degré d'utilisation extérieure : Le nombre de classes hors d'un package qui dépendent d'une classe dans le package
Is-Interface	- 0 pour classe - 1 pour interface
SUMAK	- Nombre de liens entre les méthodes et les attributs à l'intérieur de la classe
Num_Methods	- Nombre de methods
Num_Methods2	- Nombre de méthodes appelant au moins un attribut de la classe
Num_Remote_ Methods	- Nombre de méthodes appelant des méthodes des autres classes.
Num_Fields	- Nombre des attributs
Num_Pub_Fields	- Nombre des attributs publiques
INH_Level	- Nombre des niveaux d'héritage
LCOM **Lack of Cohesion of Methods**	- Une mesure de la cohésion d'une classe. - Plus le nombre est petit est plus la classe est cohérente, un nombre proche de un indique que la classe pourrait être découpée en sous-classe.
LCOM2	- cohésion des méthodes appelant au moins un attribut de la classe.
Abstract Ratio	- Le rapport d'abstraction correspond au nombre de classes abstraites et d'interfaces divisés par le nombre total de classes dans un package. - Cela nous donne donc le pourcentage de classes abstraites par package
Weighted Methods	- La somme de la complexité cyclomatique de McCabe pour toutes les méthodes de la classe. - La complexité cyclomatique c'est-à-dire le nombre de chemins possibles à l'intérieur d'une méthode, condition, opérateur ternaire, … - Il ne faut pas que ce nombre soit trop grand pour ne pas compliquer les tests et la compréhensibilité de la méthode.
Num-Parents	- Nombre totale des super-classes d'une classe.
Num-Children	Le nombre total de sous-classes directes d'une classe
CFAN-In	- class fan-in (nombre de classes appelant les méthodes appelant les méthodes de cette classe)
CFAN-Out	- class fan-out (nombre de classes appelé par cette classe)
Response	- Class response (nombre des méthodes et des remote methods)
OO-Complexity	- Indice d'OO complexity (entier compris entre 0 et 3)
Code Lines	- Nombre total de lignes de code. Les lignes blanches et les commentaires ne sont pas comptabilisés
Comment Lines	- Nombre total de ligne de commentaires.

Figure 4.1 **—** Métriques de classe.

4.1.2 Métriques de méthode

Les métriques de méthode nous renseignent sur les mesures de la qualité de la méthode contenue dans la classe Java.

Code lines	- Nombre total de lignes de code dans la méthode. Les lignes blanches et les commentaires ne sont pas comptabilisés.
Comment lines	- Nombre total de ligne de commentaires.
Coupling	Le couplage est une métrique indiquant le niveau d'interaction entre deux ou plusieurs objets.
Fan in	- Nombre total des objets appelant cette méthode.
Fan out	- Nombre total des objets appelés dans cette méthode.
Cyclomatic	- Complexité cyclomatique pour une méthode (nombre de points de décision dans cette méthode)
Parameters	- le nombre de paramètres de cette méthode (s'il y existe).
Nb recursive	- Nombre des appels récursifs pour cette méthode.
Cm ratio	- rapport : commentaire/ ligne de code
H_PGM_length	- Cette métrique et les deux suivantes sont des métriques d'**Halstead.** - Il s'agit d'une métrique textuelle, elle vise à évaluer la taille d'un programme et l'effort nécessaire en fournissant une alternative au calcul du nombre de lignes de code source. - La base des mesures est fournie par le vocabulaire utilisé. On évalue le nombre d'opérateurs et d'opérandes. - n1 = nombre d'opérateurs uniques - n2 = nombre d'opérandes uniques - N1 = nombre total d'apparition de ces opérateurs - N2 = nombre total d'apparition de ces opérandes. la longueur calculée du programme peut être approximée par $Nc = n1 \log2 n1 + n2 \log2 n2$
H_PGM_vocabulary	Taille du vocabulaire $n=n1+n2$ Longueur du vocabulaire $N=N1+N2$
H volume	- le volume du programme: $V= N\log2 n$
Integration	- Une mesure de l'interaction entre les méthodes dans un programme.
Essential	- La mesure du code qui contient des constructions non structurées. Si « Essential complexity » est trop élevée c'est qu'on a un code difficile à le comprendre (il vaut mieux le refactoriser ou même le restructurer). - Le seuil couramment utilisé est de 4. Lorsque « essential complexity » donné est supérieure à 4, la probabilité de maintenir difficilement le code est beaucoup plus élevée.
Long lines	- Nombre des longues lignes dans la méthode.
Code depth	- La profondeur du code d'une méthode.

Figure 4.2 — Métriques de méthode.

4.2 Evaluation des métriques

4.2.1 Etude de la corrélation

L'évaluation a porté sur la corrélation. Le coefficient de corrélation sert avant tout à caractériser une relation linéaire positive ou négative. Plus il est proche de 1 (en valeur absolue), plus la relation est forte.
Nous avons calculé le coefficient de corrélation entre les différentes métriques de classe ainsi que pour les métriques de méthode. La formule est la suivante :

$$r_p = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

On suppose qu'on a les tableaux de valeurs suivants : X ($x_1,..., x_n$)et Y($y_1,..., y_n$)

et pour chacune des deux séries de métriques. Alors, pour connaître le coefficient de corrélation liant ces deux séries, on applique la formule suivante :

$$r_p = \frac{\sum_{i=1}^{N}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

Si r vaut 0, les deux courbes ne sont pas corrélées. Les deux courbes sont d'autant mieux corrélées que r est loin de 0 (proche de -1 ou 1).
L'analyse montre l'existence d'une relation de corrélation entre certaines métriques. Il importe donc de bien examiner les propriétés de la matrice de corrélation avant de la soumettre à l'analyse.
Nous avons considéré que deux métriques sont corrélées positivement si r >=0.80 et négativement si r <= -0.80, nous avons jugé que ces deux valeurs sont les plus significatives pour cette étude.

4.2.2 Analyse des résultats

Les tableaux de corrélation (voir Annexe), obtenus avec le logiciel de statistique R, présentent les matrices de corrélation respectives de métriques de classes et de méthodes. Dans le cas des métriques de classe, le tableau est donc constitué de 441 cellules et pour celui des métriques de méthode 256 cellules.

Certains des coefficients sont particulièrement petits, par exemple entre le CE et le CA (0.01) ou entre Num_Fields et Num_Children(-0.04) pour les métriques de classe, et entre Fan_In et Cm_Ratio (0.01) ou entre Coupling et Fan_out (-0.01) pour les métriques de méthodes. Il n ' y a aucun intérêt pour s'intéresser à ces métriques qui sont indépendants, il faut nécessairement que les variables soient intercorrélées.
Les deux matrices de corrélation comportent aussi un certain nombre de coefficients de taille intéressante entre CE et Cfan_Out (0.93), Lcom et Is_Interface (-0.80) pour les métriques de classe et entre Code_Lines et H_Volume (0.95) pour les métriques de méthodes.
Les corrélations sont présentées dans les 2 tableaux en Annexe, les corrélations positives sont marquées en gris, et les corrélations négatives sont marquées en rouge.
Exemple de corrélation sous forme graphique de différentes métriques (voir figure 4.3) :

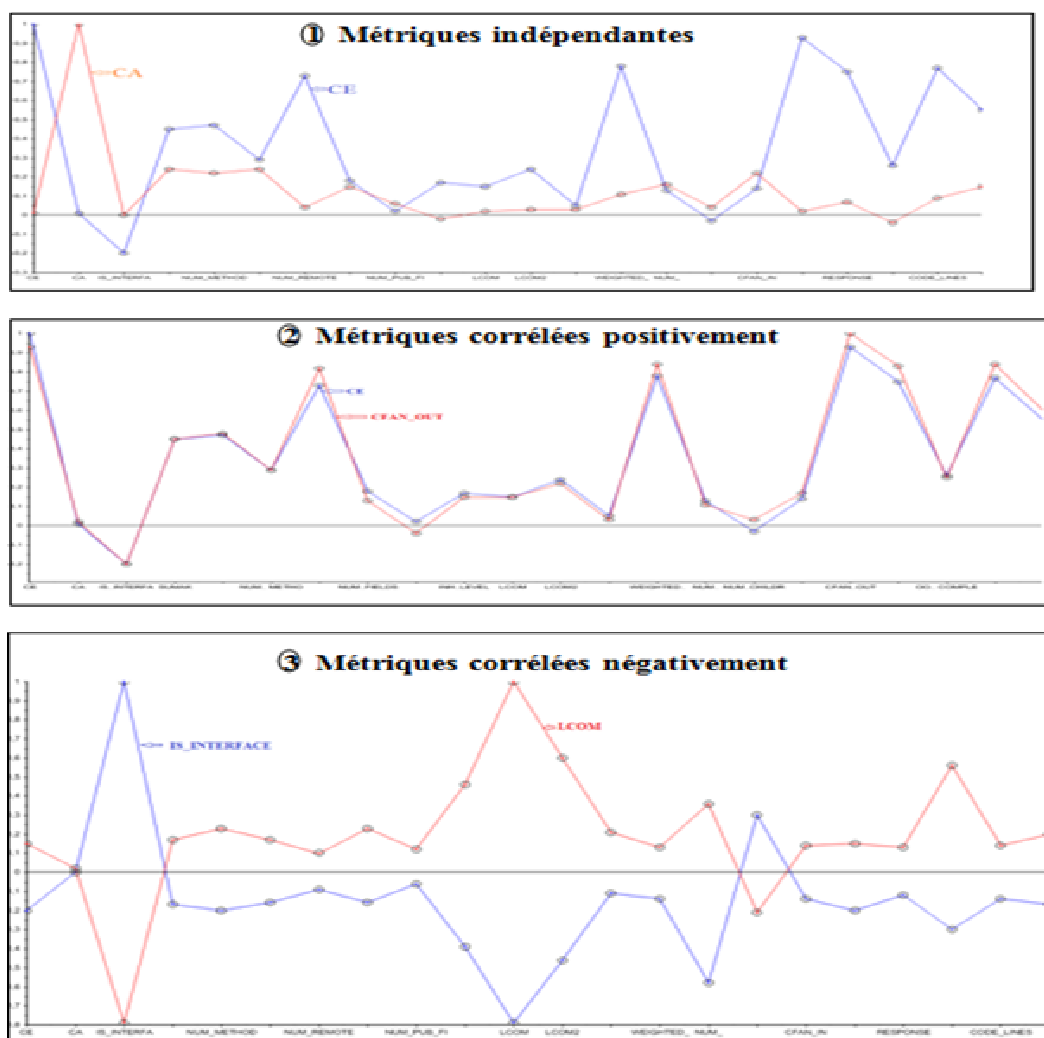


Figure 4.3 — Corrélations possibles entre les métriques.

- La première courbe : nous avons pris comme exemple les 2 métriques CE et CA,(coefficient de corrélation = 0.01) on remarque qu'elles n'ont pas de relations entre eux. Elles sont indépendantes l'une de l'autre.
- La deuxième courbe : Il y a une forte liaison entre les CE et CFAN_OUT (coefficient de corrélation=0.93). Nous constatons la grande ressemblance de la corrélation de ses deux métriques. Ces derniers sont corrélées entre eux. Deux métriques liées dont la corrélation est positive varient dans le même sens, si l'une s'améliore l'autre s'améliore aussi. Par conséquent, on peut affirmer que le fait d'améliorer CE par exemple, CFAN_OUT évolue nécessairement.
- La troisième courbe : il existe une relation négative entre LCOM et is_Interface (coefficient de corrélation = 0.80). Nous pouvons remarquer que les 2 courbes sont opposées. Donc l'amélioration de l'une des métriques corrélées négativement, entraine une baisse certaine pour l'autre.

Cette étude nous mène à dire, que l'une des raisons qui ont causé la régression de la robustesse est la corrélation négative.

Conclusion

Cette étude empirique a été menée afin d'identifier les relations entre les métriques, évaluer la corrélation entre eux et estimer les changements de la qualité des logiciels.

Ainsi , notre étude de la corrélation met en évidence son importance pour l'étude des métriques

Conclusion

Les systèmes logiciels doivent obligatoirement subir des modifications et afin de répondre à l'évolution des exigences. Aujourd'hui, les bonnes pratiques est parmi les techniques les plus importantes pour améliorer la qualité du logiciel. Cette qualité peut être mesurée en utilisant plusieurs métriques.

Nous nous sommes intéressés, dans ce travail, au développement d'une bonne pratique permettant d'aider les concepteurs, dans un contexte industriel bien défini (Agrostar), lors de la phase de la modélisation afin d'améliorer la qualité du code produit. Pour cela, nous avons utilisé le langage GooMod développé par l'équipe SE du Laboratoire Valoria.

Pour aboutir à ces résultats, nous avons tout d'abord commencé par établir une étude approfondie les métriques et leurs liens avec les bonnes pratiques. Ensuite, nous avons présenté les différentes règles que devra satisfaire le modèle conceptuel du logiciel. Finalement, nous avons abordé l'étape de réalisation où nous avons traduit les étapes de bonnes pratiques en une implémentation via GooMod.

Nous avons examiné les impacts des bonnes pratiques appliquées. Les résultats indiquent un changement significatif. Plus précisément, il semble que les bonnes pratiques ont provoqué une augmentation non négligeable au niveau des trois facteurs de qualité : transferabilité, changeabilité et sécurité. Par contre, cela à causer une régression au niveau de la robustesse. Cette dernière constatation nous a mené à étudier la corrélation entre les différentes métriques de classe et de méthode.

Nous sommes, maintenant, convaincus que les différentes métriques ne sont pas toujours compatibles et qu'il est nécessaire de trouver des compromis. Dans tous les cas, les objectifs de qualité doivent être définis pour chaque logiciel, et la qualité du logiciel doit être contrôlée par rapport à ces objectifs.

Bibliographie

[Alikacem et Sahraoui, 2006] *Alikacem, E. et Sahraoui, H. (2006).* "Generic metric extraction framework". In Proceedings of IWSM/MetriKon'2006. [Ambler, 2010] Ambler, S.W. : "The Elements of UML(TM) 2.0 Style". Cambridge University Press, New York, NY, USA ; 2005.

[Ambler, 2010] *Ambler, S.W.* "The Elements of UML(TM) 2.0 Style". Cambridge University Press, New York, NY, USA ; 2005

[Bansiya et al. , 1997] *] J. Bansiya and C. Davis,* "Automated Metrics for Object-Oriented Development," Dr. Dobb's J., vol. 272, pp. 42-48, Dec. 1997.

[Basili et al. ,1996] *Basili, V. R., Briand, L. C., et Melo, W. L. 1996.*"A validation of object-oriented design metrics as quality indicators". IEEE Trans. Softw. Eng., 22(10) :751-761.

[Beyer et al., 2005] *Beyer, D., Noack, A., et Lewerentz, C. (2005).* "Efficient relational calculation for software analysis". IEEE Trans. Softw. Eng., 31(2) :137-149.

[Boehm,1981] *Boehm B.W.*, "Software Engineering Economics, Englewood Cliffs, NJ, Prentice-Hall Inc., 1981

[Brito et al. , 1994] *Fernando Brito e Abreu and Rogerio Carapuça.* "Object-Oriented Software Engineering : Measuring and Controlling the Development Process". In Proc. of the 4th Int. Conf. on Software Quality (ASQC), McLean, VA, USA, 1994.

[Chidamber et al., 1994] *Chidamber S.R., Kemerer C.F.* : "A metrics suite for object oriented design" ; IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994

[DeMarco, 1982] *Tom De Marco* - "Controlling Software Projects : management, measurement et estimation" - Englewood Cliffs, page 3, NJ - Prentice Hall, 1982.

[IEEE, 1998] *IEEE "Software et Systems Engineering Standards Collection"*, Software Engineering Standards Committee of the IEEE Computer Society Institute of Electrical and Electronics Engineers

[ISO, 2000] *ISO 9000 :2000*"Quality management systems Fundamentals and vocabulary" Geneva, Switzeriand : Intemafional Organization for Standardization

[ISO, 2001] *ISO. 2001c.* "Technologies de l'information- Qualites de produits logiciels Partie 1 : Modele de qualite." ISO/IEC 9126-1, Geneva, Switzeriand : Intemafional Organization for Standardization

[LeGloahec et al. ,2009] *Vincent Le Gloahec, Regis Fleurquin, Salah Sadou* "Formalisation de bonnes pratiques dans les procédés de développement logiciels". 5 éme journée de l'ingénierie dirigée par les modèles (IDM 2009), Nancy France

[LeGloahec et al., 2010a] *Vincent Le Gloahec,* Regis Fleurquin, and Salah Sadou ; "Good Architecture = Good (ADL + Practices)" 6th International Conference on the Quality of Software Architectures (QoSA'10) 6093 167-182 ; 2010

[Le Gloahec et al.,2010b] *Vincent Le Gloahec, Regis Fleurquin, and Salah Sadou ;* "Good Practices as a Quality-Oriented Modeling Assistant" QSIC'10 : Proceedings of the 10th International Conference on Quality Software (2010) 345-348

[Marinescu et al., 2005] *Marinescu, C., Marinescu, R., et Gîrba, T.* (2005). "Towards a simplified implementation of object-oriented design metrics". In IEEE METRICS, page 11.

[Masud et al. , 2008] *Md. Raihan Masud, M. M. A. Hashem, Md. Abul Khaer* "Normalization Approach for Metric Based Software Quality Measurement" ; Proceedings of the International Conference on Computer and Communication Engineering 2008

[Mendling et al., 2010] *J. Mendling a ;H.A. Reijers W.M.P. van der Aalst,* "Seven Process Modeling Guidelines (7PMG)". Humboldt University, Unter den Linden 6, 10099 Berlin, Germany Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands , 2010

[Monperrus, 2008] *Martin Monperrus* "La mesure des modèles par les modèles : une approche générative". IFSIC Université de Rennes I, 2008

[Oman et al. ,1997] *Oman, Paul, et Shari Lawrence Pfleeger.* 1997. "Applying software metrics". Etats-Unis : Los Alamitos, Calif : IEEE Computer Society Press, 321 p.

[Perez et al.,2007] *Gonzalez-Perez, C., Henderson-Sellers, B.* : "Modelling software development methodologies : A conceptual foundation". Journal of Systems and Software 80(11) 2007

[Rolland, 2007] *Rolland, C.*, (2007) Capturing System Intentionality with Maps . Conceptual Modelling in Information Systems Engineering, pages 141-158. Sutcliffe, A., (2003) Scenario-based.

[Tian , 1998] *J. Tian,* "Early Measurement and Improvement of Software Quality", compsac, pp.196, 22nd International Computer Software and Application Conference, 1998

[Wagner et al. , 2008] *Wagner S., Deissenboeck F., Aichner M.* : "An Evaluation of Two Bug Pattern Tools for Java", Proceedings of the IEEE International Conference on Software

Testing, Verification, and Validation (ICST 2008), April 9-11, 2008, Lillehammer, Norwegen, IEEE Computer Society Press, 2008

[Wohlin, 2007] *Wohlin, C.* (2007). An analysis of the most cited articles in software engineering journals. Inf. Softw. Technol., 49(1) :2-11.

Annexes

ANNEXE A

Tableaux

	CE	CA	IS_INTERFACE	SUMAK	NUM_METHODS	NUM_METHODS2	NUM_REMOTE_METHODS	NUM_FIELDS	NUM_PUB_FIELDS	INH_LEVEL	LCOM	LCOM2	ABSTRACT_RATIO	WEIGHTED_METHODS	NUM_PARENTS	NUM_CHILDREN	CFAN_IN	CFAN_OUT	RESPONSE	OO_COMPLEXITY	CODE_LINES	COMMENT_LINES
CE	1,00	0,01	-0,20	0,45	0,47	0,29	0,73	0,18	0,02	0,17	0,15	0,24	0,05	0,78	0,13	-0,03	0,14	0,93	0,75	0,26	0,77	0,55
CA	0,01	1,00	0,00	0,24	0,22	0,24	0,04	0,15	0,06	-0,02	0,02	0,03	0,03	0,11	0,16	0,04	0,22	0,02	0,07	-0,04	0,09	0,15
IS_INTERFACE	-0,20	0,00	1,00	-0,17	-0,20	-0,16	-0,09	-0,16	-0,06	-0,39	-0,79	-0,46	-0,11	-0,14	-0,58	0,30	-0,14	-0,20	-0,12	-0,30	-0,14	-0,17
SUMAK	0,45	0,24	-0,17	1,00	0,88	0,90	0,41	0,56	0,09	-0,12	0,17	0,32	0,05	0,66	0,20	-0,03	0,40	0,45	0,52	0,12	0,64	0,76
NUM_METHODS	0,47	0,22	-0,20	0,88	1,00	0,94	0,43	0,50	0,06	-0,03	0,23	0,36	0,02	0,71	0,20	-0,03	0,46	0,48	0,55	0,26	0,65	0,85
NUM_METHODS2	0,29	0,24	-0,16	0,90	0,94	1,00	0,27	0,54	0,07	-0,15	0,17	0,31	0,00	0,54	0,19	-0,04	0,43	0,29	0,40	0,14	0,50	0,76
NUM_REMOTE_METHODS	0,73	0,04	-0,09	0,41	0,43	0,27	1,00	0,12	-0,01	0,07	0,10	0,13	0,04	0,88	0,03	0,01	0,09	0,82	0,99	0,20	0,92	0,64
NUM_FIELDS	0,18	0,15	-0,16	0,56	0,50	0,54	0,12	1,00	0,82	-0,07	0,23	0,24	0,50	0,26	0,11	-0,04	0,24	0,13	0,19	0,17	0,31	0,45
NUM_PUB_FIELDS	0,02	0,06	-0,06	0,09	0,06	0,07	-0,01	0,82	1,00	-0,02	0,12	0,04	0,65	0,02	-0,03	-0,02	0,03	-0,04	0,00	0,11	0,05	0,05
INH_LEVEL	0,17	-0,02	-0,39	-0,12	-0,03	-0,15	0,07	-0,07	-0,02	1,00	0,46	0,20	-0,02	0,04	0,24	-0,12	-0,08	0,15	0,06	0,63	0,02	-0,03
LCOM	0,15	0,02	-0,8	0,17	0,23	0,17	0,10	0,23	0,12	0,46	1,00	0,60	0,21	0,13	0,36	-0,21	0,14	0,15	0,13	0,56	0,14	0,20
LCOM2	0,24	0,03	-0,46	0,32	0,36	0,31	0,13	0,24	0,04	0,20	0,60	1,00	0,01	0,20	0,32	-0,13	0,21	0,22	0,18	0,32	0,21	0,31
ABSTRACT_RATIO	0,05	0,03	-0,11	0,05	0,02	0,00	0,04	0,50	0,65	-0,02	0,21	0,01	1,00	0,05	-0,02	-0,01	0,10	0,03	0,04	0,19	0,06	0,02
WEIGHTED_METHODS	0,78	0,11	-0,14	0,66	0,71	0,54	0,88	0,26	0,02	0,04	0,13	0,20	0,05	1,00	0,08	0,01	0,24	0,84	0,92	0,28	0,97	0,81
NUM_PARENTS	0,13	0,16	-0,58	0,20	0,20	0,19	0,03	0,11	-0,03	0,24	0,36	0,32	-0,02	0,08	1,00	-0,17	0,17	0,11	0,06	0,05	0,08	0,18
NUM_CHILDREN	-0,03	0,04	0,30	-0,03	-0,03	-0,04	0,01	-0,04	-0,02	-0,12	-0,21	-0,13	-0,01	0,01	-0,17	1,00	0,08	0,03	0,01	-0,05	-0,01	-0,01
CFAN_IN	0,14	0,22	-0,14	0,40	0,46	0,43	0,09	0,24	0,03	-0,08	0,14	0,21	0,10	0,24	0,17	0,08	1,00	0,17	0,16	0,11	0,20	0,38
CFAN_OUT	0,93	0,02	-0,20	0,45	0,48	0,29	0,82	0,13	-0,04	0,15	0,15	0,22	0,03	0,84	0,11	0,03	0,17	1,00	0,83	0,25	0,84	0,60
RESPONSE	0,75	0,07	-0,12	0,52	0,55	0,40	0,99	0,19	0,00	0,06	0,13	0,18	0,04	0,92	0,06	0,01	0,16	0,83	1,00	0,23	0,95	0,73
OO_COMPLEXITY	0,26	-0,04	-0,30	0,12	0,26	0,14	0,20	0,17	0,11	0,63	0,56	0,32	0,19	0,28	0,05	-0,05	0,11	0,25	0,23	1,00	0,25	0,26
CODE_LINES	0,77	0,09	-0,14	0,64	0,65	0,50	0,92	0,31	0,05	0,02	0,14	0,21	0,06	0,97	0,08	-0,01	0,20	0,84	0,95	0,25	1,00	0,78
COMMENT_LINES	0,55	0,15	-0,17	0,76	0,85	0,76	0,64	0,45	0,05	-0,03	0,20	0,31	0,02	0,81	0,18	-0,01	0,38	0,60	0,73	0,26	0,78	1,00

Figure A.1 — Corrélation entre les métriques de classe.

	CODE_ LINES	COMMENT _ LINES	COUPLING	FAN_IN	FAN_OUT	CYCLOMA- TIC	PARAME- TERS	NB_ RECURSIVE	CM_ RATIO	H_PGM_ LENGTH	H_PGM_ VOCABULAR Y	H_VOLUME	INTEGRA- TION	ESSENTIAL	LONG_ LINES	CODE_ DEPTH
CODE_LINES	1,00	0,67	-0,02	-0,02	0,71	0,76	0,21	0,03	-0,21	0,97	0,88	0,95	0,75	0,17	0,81	0,58
COMMENT_ LINES	0,67	1,00	-0,01	-0,01	0,51	0,61	0,11	0,02	0,29	0,64	0,60	0,64	0,60	0,18	0,67	0,42
COUPLING	-0,02	-0,01	1,00	0,99	-0,01	0,00	-0,01	0,00	0,01	-0,02	-0,01	-0,02	-0,01	0,00	-0,01	-0,01
FAN_IN	-0,02	-0,01	0,99	1,00	-0,01	0,00	-0,01	0,00	0,01	-0,02	-0,01	-0,02	-0,01	0,00	-0,01	-0,01
FAN_OUT	0,71	0,51	-0,01	-0,01	1,00	0,67	0,20	0,04	-0,20	0,78	0,85	0,79	0,67	0,11	0,79	0,57
CYCLOMATIC	0,76	0,61	0,00	0,00	0,67	1,00	0,16	0,04	-0,18	0,72	0,72	0,70	0,99	0,21	0,76	0,68
PARAMETERS	0,21	0,11	-0,01	-0,01	0,20	0,16	1,00	0,10	-0,14	0,19	0,29	0,18	0,16	0,04	0,18	0,20
NB_ RECURSIVE	0,03	0,02	0,00	0,00	0,04	0,04	0,10	1,00	-0,02	0,02	0,05	0,02	0,03	0,00	0,02	0,06
CM_RATIO	-0,21	0,29	0,01	0,01	-0,20	-0,18	-0,14	-0,02	1,00	-0,19	-0,30	-0,17	-0,18	-0,04	-0,12	-0,27
H_PGM_ LENGTH	0,97	0,64	-0,02	-0,02	0,78	0,72	0,19	0,02	-0,19	1,00	0,88	0,99	0,72	0,12	0,88	0,52
H_PGM_ VOCABULARY	0,88	0,60	-0,01	-0,01	0,85	0,72	0,29	0,05	-0,30	0,88	1,00	0,86	0,71	0,19	0,78	0,70
H_VOLUME	0,95	0,64	-0,02	-0,02	0,79	0,70	0,18	0,02	-0,17	0,99	0,86	1,00	0,70	0,11	0,89	0,48
INTEGRATION	0,75	0,60	-0,01	-0,01	0,67	0,99	0,16	0,03	-0,18	0,72	0,71	0,70	1,00	0,16	0,76	0,66
ESSENTIAL	0,17	0,18	0,00	0,00	0,11	0,21	0,04	0,00	-0,04	0,12	0,19	0,11	0,16	1,00	0,13	0,21
LONG_LINES	0,81	0,67	-0,01	-0,01	0,79	0,76	0,18	0,02	-0,12	0,88	0,78	0,89	0,76	0,13	1,00	0,50
CODE_DEPTH	0,58	0,42	-0,01	-0,01	0,57	0,68	0,20	0,06	-0,27	0,52	0,70	0,48	0,66	0,21	0,50	1,00

***Figure A.2* —** Corrélation entre les métriques de méthode.

Transferability

	Critere	Ponderation
Architecture	Object-level Dependencies	3
Complexity	Algorithmic and Control Structure Complexity	9
Complexity	Dynamic Instantiation	4
Complexity	Empty Code	1
Complexity	OO Inheritance and Polymorphism	5
Complexity	SQL Queries	8
	Dead code (static)	3
Documentation	Automated Documentation	6
Documentation	Bad Comments	8
Documentation	Naming Convention Conformity	8
Documentation	Style Conformity	6
Documentation	Volume of Comments	8
Programming Practices	File Organization Conformity	1
Programming Practices	OO Inheritance and Polymorphism	3
Programming Practices	Structuredness	3
Volume	Number of Components	3
Volume	Number of LOC	3

Changeability

	Critere	Pondération
Architecture	Architecture Models Automated Checks	8
Architecture	Multi-Layers and Data Access	8
Architecture	OS and Platform Independence	3
Architecture	Object-level Dependencies	7
Complexity	Algorithmic and Control Structure Complexity	3
Complexity	Dynamic Instantiation	5
Complexity	Empty Code	1
Complexity	Functional Evolvability	7
Complexity	OO Inheritance and Polymorphism	3
Complexity	SQL Queries	4
	Dead code (static)	1
Documentation	Naming Convention Conformity	3
Documentation	Volume of Comments	3
Programming Practices	Modularity and OO Encapsulation Conformity	8
Programming Practices	Structuredness	7

Robustness

	Critére	Ponderation
Architecture	Architecture Models Automated Checks	6
Architecture	Multi-Layers and Data Access	6
Architecture	OS and Platform Independence	4
Architecture	Object-level Dependencies	7
Architecture	Reuse	4
Complexity	Algorithmic and Control Structure Complexity	7
Complexity	Dynamic Instantiation	8
Complexity	OO Inheritance and Polymorphism	4
Complexity	SQL Queries	6
Complexity	Technical Complexity	5
	Dead code (static)	2
Programming Practices	Error and Exception Handling	10
Programming Practices	OO Inheritance and Polymorphism	4
Programming Practices	Structuredness	3
Programming Practices	Unexpected Behavior	10
Volume	Number of Components	1

Performance

	Critére	Ponderation
Complexity	Dynamic Instantiation	7
Complexity	SQL Queries	5
Efficiency	Expensive Calls in Loops	10
Efficiency	Memory, Network and Disk Space Management	10
Efficiency	SQL and Data Handling Performance	9

Security

	Critére	Ponderation
Architecture	Architecture Models Automated Checks	8
Architecture	Multi-Layers and Data Access	8
Architecture	OS and Platform Independence	3
Efficiency	Memory, Network and Disk Space Management	5
Programming Practices	Error and Exception Handling	10
Programming Practices	Unexpected Behavior	10
Secure Coding	API Abuse	8
Secure Coding	Encapsulation	5
Secure Coding	Input Validation	10
Secure Coding	Time and State	4
Secure Coding	Weak Security Features	4