Évaluation de l'impact des bonnes pratiques sur la qualité du logiciel: une étude de cas industriel

Salma Hamza

PRINCE Lab., University of Sousse, Sousse, Tunisie
salma.hamza@prince.rnu.tn

Abstract

The capitalization of good practices is a controversial issue. Supporters claim that it helps to increase the productivity of the practitioners who employ it, and to improve the internal quality of the code, which facilitates its understanding and its reusability. However, there is no empirical evidence to support such assumptions and therefore, its use is still extremely rare. Thereby, it is important to study the actual impact of these good practices on the quality of the produced code. The aim of this paper is to validate/invalidate the claims that using good practices improves software quality. For that, we conduct a case study in an industrial environment whose objective is to determine the variation of the code quality using a good Modeling practices's specification language. We evaluate its effect on five quality factors. The results indicate that the use of good practices does not improve all quality factors.
Keywords— Good Practices, Variation of the quality, Case Study

Résumé

La capitalisation des bonnes pratiques est un sujet controversé. Les partisans affirment que cela contribue à accroitre la productivité des praticiens qui l'emploient, ainsi que la qualité interne du code, ce qui facilite sa compréhension et sa réutilisation. Cependant, il n'y a pas de preuve empirique pour soutenir de telles hypothèses et, par conséquent, son utilisation est encore extrêmement rare. Ainsi, il est important d'étudier l'impact réel de ces bonnes pratiques sur la qualité du code produit. L'objectif de ce papier est de valider/invalider les affirmations selon lesquelles l'utilisation des bonnes pratiques améliore la qualité du logiciel. De ce fait, nous allons mener une étude de cas dans un environnement industriel dont l'objectif est de déterminer la variation de la qualité du code en utilisant un langage de spécification de bonnes pratiques de modélisation GooMod. Nous avons évalué son effet sur cinq attributs de qualité. Les résultats indiquent que l'utilisation de bonnes pratiques n'améliore pas tous les attributs de qualité.
Mots clés— Bonnes Pratiques, Variation de la Qualité, Étude de Cas

Table des matières

1 Introduction

Les bonnes pratiques ont été préconisées depuis longtemps afin d'améliorer la qualité et accroître la productivité et l'efficacité du développement de logiciels. En effet, elles n'ont cessé de susciter un intérêt grandissant au sein du monde académique : différents processus, modèles et outils de capitalisation ont été mis en place afin de permettre aux entreprises de préserver leur expertise [3, 8, 10, 11]. Néanmoins, les bonnes pratiques ne sont généralement pas appliquées. En effet, des difficultés majeures surgissent pour les introduire dans la pratique : les praticiens croient souvent que les bonnes pratiques sont complexes, coûteuses et nécessitent des compétences techniques élevées et beaucoup de temps et donc difficiles à appliquer. En outre, il n'est pas souvent clair de savoir comment les utiliser afin d'améliorer les attributs de qualité spécifiques qui sont des indicateurs pour une bonne conception. La pertinence des bonnes pratiques est, de ce fait, tout simplement remise en question. Si l'on veut contrôler les impacts que les bonnes pratiques engendrent et prouver leur utilité à la communauté, il est nécessaire d'apporter des preuves empiriques.

Cet article donne une première étape vers cet objectif, en proposant une étude empirique afin d'assimiler la relation entre les bonnes pratiques et la structure interne du code source et son impact sur la structure du programme. L'hypothèse de cette étude est que les bonnes pratiques ont un impact positif sur la qualité. Notre objectif est de quantifier cet impact sur la qualité globale des systèmes pour confirmer ou refuser l'hypothèse. Nous avons effectué cette étude à l'entreprise STEF-TFE en utilisant un projet développé avec le langage de programmation Java. Ce papier est organisé comme suit : la première partie donne un aperçu des bonnes pratiques et du langage de spécification utilisé dans cette étude. La deuxième partie était destinée à comprendre comment les bonnes pratiques s'appliquent à un système de logiciel industriel. La troisième partie de cette étude décrit l'impact de ces bonnes pratiques sur la structure du code. Cette partie avait également pour objectif d'analyser l'impact des bonnes pratiques sur les métriques de qualité axées sur l'objet et de discuter les raisons de variations de résultats.

2 Concepts et définitions

Cette section décrit brièvement les bonnes pratiques ainsi que le langage, utilisé dans cette étude empirique, afin de pouvoir les définir et les exécuter.

2.1 Les bonnes pratiques

Selon LeGloahec et al., les bonnes pratiques sont des techniques et des méthodes bien définies qui, grâce à l'expérience et à la recherche, ont prouvé leur efficacité et leur contribution pour aboutir à un résultat souhaité dans le développement de logiciels. Elles sont capitalisées et issues de différentes sources : des sources externes et des sources internes à l'entreprise [6]. Elles peuvent être exprimées dans un langage de spécification du domaine (DSL). En effet, ce langage est conçu pour répondre à une problématique spécifique. 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. Différents langages de spécification des bonnes pratiques existent dans la littérature. Ces langages peuvent fournir des outils essentiels et complémentaires à la fois : tels que les processus de modélisation [4], les mesures et les styles [1].

2.2 GooMod : Langage de modélisation des bonnes pratiques

Pour pouvoir définir et exécuter les bonnes pratiques nous avons utilisé un langage de spécification des bonnes pratiques de modélisation nommé GooMod[1](Good Modeling Practices). Ce langage est issu des travaux de l'équipe ArchWare de l'IRISA-UBS [7] et est développé en utilisant la plateforme Eclipse. Il fournit les propriétés nécessaires à la description des meilleures pratiques de conception et nous permet de nous assurer de son respect lors de la production d'un code. Ce langage est associé à tout type de DSL 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. Ce langage est utilisé pour établir les conventions de codage dès la phase de la modélisation. Ces conventions ont été définies 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.

3 Étude de cas

L'hypothèse de cette étude est que les bonnes pratiques ont un impact positif sur la qualité. Notre objectif est de quantifier cet impact sur la qualité globale des systèmes pour valider ou rejeter l'hypothèse. Nous testons cette hypothèse empiriquement avec un projet de logiciel développé à STEF-TFE. Le langage de programmation utilisé était Java. Le logiciel développé se compose de 400 classes Java et un total de 5000 méthodes.

Avant de décrire nos bonnes pratiques, il pourrait vous être utile de comprendre comment nous les avons élaborées et appliquées. Pour aider les développeurs et les architectes à appliquer les bonnes pratiques, un processus est mis en place. Ce processus, avec GooMod, peut être appliqué pour différents types de projets : i) développement d'objet à partir de zéro ou ii) la ré-ingénierie, lorsque l'objectif est d'améliorer un système existant. Dans notre étude, nous avons appliqué la ré-ingénierie puisque le projet existe déjà.

3.1 Processus de développement d'un logiciel avec les bonnes pratiques

Avant d'intégrer l'outil de bonnes pratiques, le processus de développement d'un logiciel chez STEF-TFE était basé sur l'évaluation de la qualité en utilisant les métriques internes de qualité (automatisé dans l'outil CAST). Si le développeur obtient un mauvais résultat, il est nécessaire de revenir à l'étape du codage pour améliorer la qualité. Donc, ce processus basé sur un outil de mesure, reste incapable d'améliorer réellement la qualité. Par conséquent, ce processus ne satisfait pas les responsables de qualité de STEF-TFE.

Notre objectif est d'adapter le nouvel outil de BP basé sur le langage de modélisation GooMod aux besoins du STEF-TFE afin d'avoir une meilleure qualité. La figure 1 illustre le processus d'amélioration d'un logiciel avec GooMod.

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

[1]https ://www-archware.irisa.fr/software/goomod/

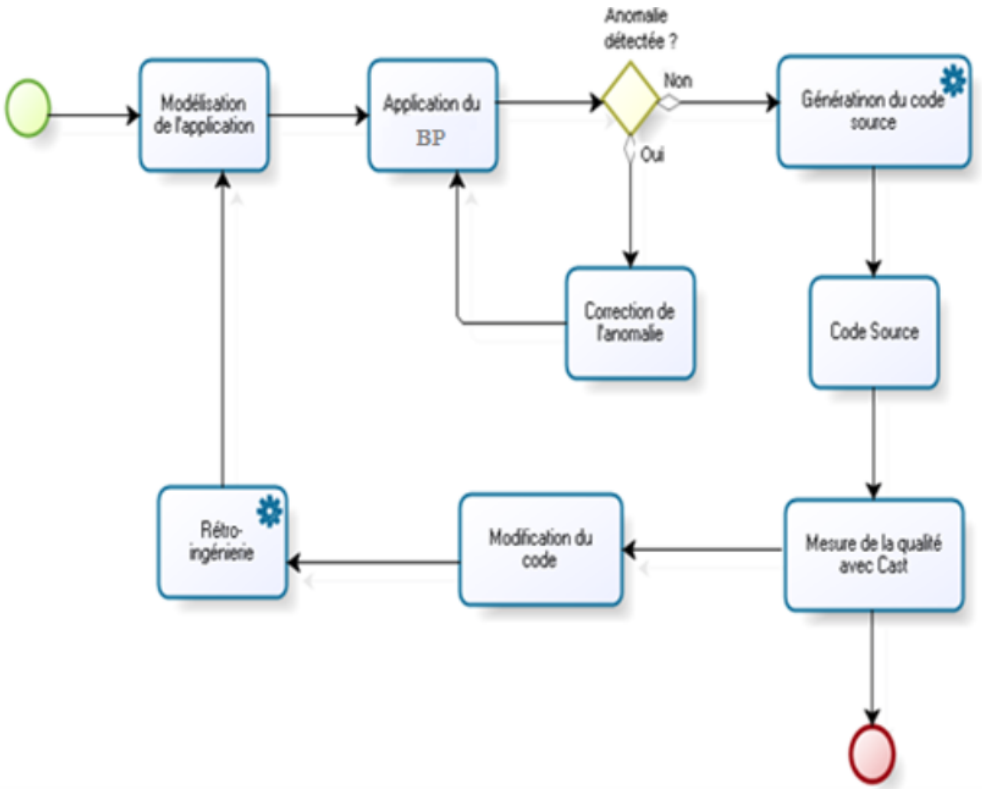

FIGURE 1 : Processus du développement d'un logiciel en utilisant les bonnes pratiques de modélisation

qualité de son code généré avec CAST. Les mesures nous permettent de savoir si la technique des bonnes pratiques améliore vraiment notre logiciel.

3.2 Modélisation des bonnes pratiques

Pour élaborer notre liste de bonnes pratiques, nous nous sommes appuyés sur le standard ISO9126 (source externe). Donc, nos bonnes pratiques ont été proposées en tant que normes générales plutôt que des prescriptions spécifiques sur l'application en question.

L'outil GooMod nous permet de représenter les différentes étapes de BP sous la forme d'un processus. Dans notre processus, nous avons défini des règles de BP, organisées sous la forme de graphe voir figure 2. Elles sont précisément définies en se basant sur le méta-modèle UML 2.0.

Nous avons défini et associé pour chacune des étapes de notre processus :

- Des pré- et/ou post-conditions : des contraintes qualitatives pour améliorer la lisibilité du code (convention du nommage), réduire la complexité du code qui est due au polymorphisme et à l'héritage et des contraintes quantitatives (beaucoup d'éléments dans un même package par exemple).
- Une liste des actions d'entrées et de sorties : les actions d'entrées sont lancées juste après la vérification de la pré-condition. Elles permettent l'initialisation de l'environnement associé

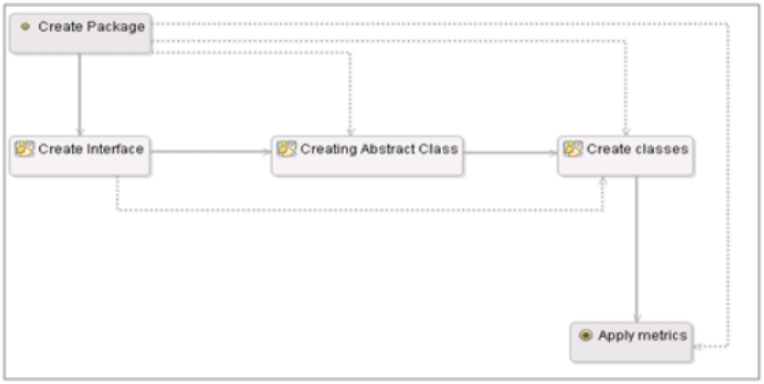

FIGURE 2 : Processus défini via GooMod

à cette étape. De même, lorsque le concepteur indique qu'il a fini de cette étape, une liste d'actions de sortie sera lancée 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.

- Un ensemble de concepts : en effet, dans une étape donnée, le concepteur est limité par des concepts. Chaque concept lui a été associé différents types d'utilisation (création, suppression, mise à jour).

3.3 Définition des bonnes pratiques

Dans cette partie, nous fournissons des extraits de toutes les étapes qui sont impliquées dans notre processus de test.

1. **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 tous les packages du modèle, qui ne respectent pas la convention.

2. **Créer des interfaces :** à cette étape, seuls les concepts d'interface, d'opération, et de liaisons, propres à cette étape, sont mis 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 définie 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 dépasse pas vingt.

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

3. **Créer des classes abstraites :** Dans les pratiques de développement chez STEF-TFE, chaque matériel est associé à une classe abstraite donc 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 définie 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. **Créer des classes :** à cette étape, les concepts de classes, d'attribut, d'opération, et de liaisons, propres à cette étape, sont mis à la 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'il y a au moins 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. **Appliquer des métriques :** nous avons consacré la dernière étape à la vérification de certains aspects que nous ne pouvons pas vérifier au cours d'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 terminé.

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

Dans cette partie, nous nous sommes appuyés sur le standard ISO 9126 et l'ensemble de métriques utilisées par STEF-TFE pour la définition d'une telle BP. Dans la partie suivante, nous allons voir l'impact de ses BP sur la qualité du logiciel.

4 Évaluation de l'impact des bonnes pratiques

Nous n'avons pas évalué la fonctionnalité du programme. En effet, les bonnes pratiques, par définition, ne modifient pas le comportement des systèmes mais plutôt elles modifient sa structure interne.

4.1 Attributs de la qualité du logiciel

Nous avons utilisé le modèle de qualité CAST pour estimer l'effet des bonnes pratiques suggérées sur les attributs de qualité. Ce modèle est basé principalement sur le standard CISQ(Consortium for IT Software Quality) [5, 12] lancé par le consortium OMG(Object Management Group). Le modèle de qualité CAST adopte l'approche de décomposition utilisée par McCall [9] et Boehm [2] communément appelé le modèle de qualité facteurs-métriques (FCM). Il permet de fournir des mesures sur un certain nombre de critères de qualité de la norme de qualité ISO/CEI 25010. Nous choisissons CAST, principalement parce qu' (1) il peut détecter les violations du bon codage et le respect des règles de programmation dans les logiciels (2) il a l'avantage de définir cinq attributs de qualité de conception de haut niveau qui peuvent être calculés en utilisant les métriques de conception de niveau inférieur. Dans notre étude, nous considérons les attributs de qualité suivants :

- Robustesse : la capacité des systèmes logiciels à réagir de manière appropriée à des conditions anormales
- Transferabilité : la capacité d'un module à être utilisé dans plus d'un programme informatique ou d'un système logiciel.
- Changeabilité : détermine la facilité et la rapidité avec laquelle une application peut être modifiée.
- Performance : évalue les caractéristiques qui affectent le comportement de réponse d'une application.
- Sécurité : la capacité d'un produit ou un système à protéger ses informations et ses données.

Quality Factor	**Quality Index Calculation**
Robustesse	=10* (Error and Exception Handling+Unexpected Behavior)+8*(Dynamic Instantiation)+7*(Object level Dependencies+Algorithmic and Control Structure Complexity)+6*(Architecture Models Automated Checks+Multi-Layers and Data Access+SQL Queries)+5*(Technical Complexity)+4*(OS and Platform Independence +Reuse+OO Inheritance and Polymorphism)+3*(Structuredness)+2*(Dead code)+1*(Number of Components)
Transferabilité	=9*(Algorithmic and Control Structure Complexity) +8*(SQL Queries + Bad Comments+ Naming convention Conformity+Volume of Comments)+ 6 *(Automated Documentation+ Style Conformity) + 5 *(OO Inheritance and Polymorphism)+ 4* (Dynamic Instantiation) +3 * (Object-level Dependencies+Dead code +Number of Components + Number of LOC+Structurdness) + 1* (Empty Code+ File Organization conformity)
Changeabilité	=8*(Architecture Models Automated Checks + Multi-Layers and Data Access + Modularity and OO Encapsulation Conformity) + 7*(Object-level Dependencies + functional Evolvability+ structuredness)+5*(Dynamic Instantiation)+ 4*SQL Queries + 3*(OS and Plateform Independence + Algorithmic and Controle Structure Complexity + OO Inheritance and Polymorphism + Naming Convention Conformity + Volume of Comments)+1* (Empty Code+ Dead Code)
Performance	=10*(Memory, Network and Disk Space Management) +9*(SQL and Data Handling Performance)+ 7*(Dynamic Instantiation +SQL Queries +10*Expensive Calls in Loops)
Sécurité	=10*(Input Validation+Error and Exception Handling+Unexpected Behavior)+8*(Architecture Models Automated Checks+Multi-Layers and Data Access+API Abuse) +5*(Memory, Network and Disk space Management+Encapsulation)+4*(Time and State+Weak Security Features)+3*(OS and Plateform Independence)

Table 1 : CAST Quality Factors

Le tableau 1 résume les attributs de qualité de CAST. Il décrit les relations entre les facteurs et les critères, de sorte que nous pouvons mesurer les facteurs en fonction des critères de la dépendance. Chaque facteur de qualité est composé de critères de niveau inférieur, tels que la structuration et la concision. Ces critères sont plus faciles à comprendre et à mesurer que les facteurs eux-mêmes, de sorte que des mesures réelles sont proposées pour eux.

4.2 Évaluation de l'impact des bonnes pratiques sur les attributs de qualité externe

L'amélioration de la qualité peut être évaluée en comparant la qualité avant et après l'application des bonnes pratiques. Par conséquent, l'amélioration d'un facteur de qualité donné (AFq) du CAST est estimée comme suit : AFq= $Fq_{i'}$-Fq_i. Où Fq_i et $Fq_{i'}$ sont les valeurs des facteurs de qualité i respectivement, avant et après les bonnes pratiques.

Dans la table 2, nous montrons les valeurs obtenues pour chaque facteur de qualité CAST avant et après les bonnes pratiques pour le système étudié de STEF-TFE. Nous avons constaté

	Transferabilité	Changeabilité	Robustesse	Performance	Sécurité
Projet (Sans BP)	2.89	2.99	3.08	2.86	2.71
Projet (Avec BP)	2.9	3	3.06	2.86	2.72

TABLE 2 : L'impact des bonnes pratiques sur les facteurs de qualité CAST

que la qualité du système augmente dans trois facteurs de qualité. Nous nous basons, ici, 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 critères.

- **Transferabilité :** Parmi les critères du facteur transferabilité, figure le critère de conformité des conventions du nommage 'Documentation Naming convention conformity'. Ce critère est amélioré de 0.09 (voir figure 3).

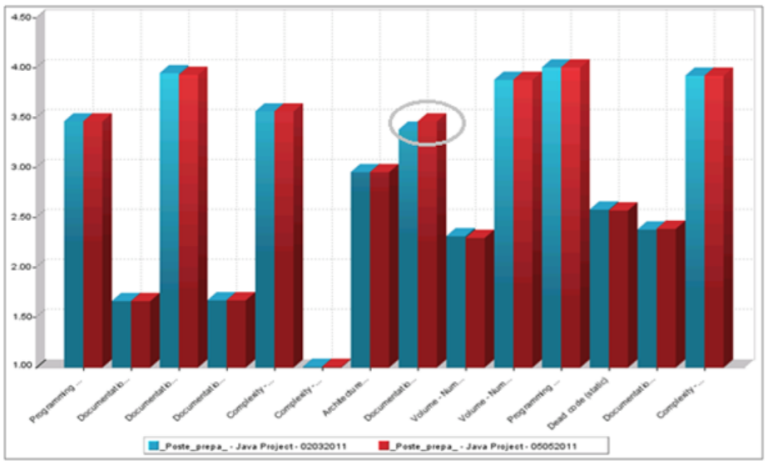

FIGURE 3 : Les critères du facteur de qualité Transferabilité

Ce critère s'est amélioré grâce à l'amélioration de la métrique convention du nommage des packages "Package naming convention"(0.42 de hausse).

- **Changeabilité :** Un accroissement a été enregistré également au niveau du facteur changeabilité du fait de l'amélioration du critère Architecture Models Automated Checks (0.05 de hausse) et ceci est du aux règles liées à la complexité qui ont été mises en place.
- **Sécurité :** Le critère Architecture Models Automated Checks existe de même dans ce facteur de qualité. Donc le fait de l'améliorer, améliore plusieurs facteurs à la fois.
- **Robustesse :** Par contre, le facteur de qualité Robustesse accuse une régression de (-0.02). Le recul observé dans la robustesse s'explique par le déclin enregistré dans le critère Volume Nombre des composants. Ce déclin s'explique par le fait que notre BP indique que la classe dépasse le nombre d'opérations, nous éclatons dans ce cas la classe en 2 classes pour suivre les bonnes pratiques.

L'analyse des résultats de notre étude révèle que, contrairement à la tradition commune, les bonnes pratiques n'influent pas toujours positivement sur les attributs de qualité. Elles n'améliorent pas toujours la qualité des systèmes dans lesquels ils sont appliqués. Dans notre étude, ils ont réduit sensiblement la robustesse. Pourquoi le facteur de robustesse a-t-il baissé ? Néanmoins, pour mieux comprendre les résultats obtenus et trouver une explication à cette question, une étude plus fine est toutefois nécessaire.

4.3 Discussion

Notre hypothèse était que les bonnes pratiques aident à produire des systèmes de bonne qualité. Il est surprenant que leurs utilisations semble diminuer la qualité. Cette constatation nous mène à étudier la corrélation entre les différentes métriques de classe. En effet, 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 critères. Chaque critère est composé d'un ensemble de métriques. Ces métriques sont appliquées directement sur le code. Notre étude, dans cette partie, porte sur les métriques et les impacts directs qu'elles peuvent avoir sur les facteurs de qualité. L'évaluation est effectuée en utilisant 21 métriques de classe. 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.

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). Ces métriques sont indépendantes, de ce fait, il n'y a aucun intérêt pour s'intéresser à ce type de dépendances.

La matrice de corrélation comporte aussi un certain nombre de coefficients de taille intéressante par exemple les deux métriques CE et CFAN-OUT sont fortement corrélées positivement entre elles (0.93). Ces deux métriques 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. Par contre, les deux métriques Lcom et Is-Interface sont fortement corrélées négativement entre elles (-0.80). 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.

5 Conclusion

Cette étude de cas résume brièvement les résultats qui peuvent être obtenus en utilisant les bonnes pratiques. Il s'agit d'une synthèse basée sur certains attributs de qualité lors d'une amélioration et ensuite une évaluation du système de l'entreprise STEF-TFE. Nous avons

présenté les différentes règles que devra satisfaire le modèle conceptuel du logiciel. Ensuite, 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 au niveau des trois facteurs de qualité : transferabilité, changeabilité et sécurité. Par contre, cela a causé une régression au niveau de la robustesse. Cette dernière constatation nous a mené à étudier la corrélation entre les différentes métriques de classe. Nous sommes 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.

Références

[1] Scott W Ambler. *The Elements of UML (TM) 2.0 Style*. Cambridge University Press, 2005.

[2] Barry W Boehm, John R Brown, and Mlity Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605, 1976.

[3] G Fragidis and K Tarabanis. From repositories of best practices to networks of best practices. In *Management of Innovation and Technology, 2006 IEEE International Conference on*, volume 1, pages 370–374. IEEE, 2006.

[4] Cesar Gonzalez-Perez and Brian Henderson-Sellers. Modelling software development methodologies : A conceptual foundation. *Journal of Systems and Software*, 80(11) :1778–1796, 2007.

[5] OMG CISQ Technical Work Groups. Cisq specifications for automated quality characteristic measures. Technical report, 2012.

[6] Vincent Le Gloahec, Regis Fleurquin, and Salah Sadou. Formalisation de bonnes pratiques dans les procédés de développement logiciels. In *5èmes Journées de l'Ingénierie dirigée par les modèles (IDM 2009)*, pages 95–100, 2009.

[7] Vincent Le Gloahec, Régis Fleurquin, and Salah Sadou. Good architecture = good (adl + practices). In *Proceedings of the 6th International Conference on Quality of Software Architectures : Research into Practice - Reality and Gaps*, QoSA'10, pages 167–182, 2010.

[8] Vincent Le Gloahec, Regis Fleurquin, and Salah Sadou. Good practices as a quality-oriented modeling assistant. In *Quality Software (QSIC), 2010 10th International Conference on*, pages 345–348. IEEE, 2010.

[9] Jim A McCall, Paul K Richards, and Gene F Walters. Factors in software quality. volume i. concepts and definitions of software quality. Technical report, 1977.

[10] Raman Ramsin and Richard F Paige. Process-centered review of object oriented software development methodologies. *ACM Computing Surveys (CSUR)*, 2008.

[11] Bradley Schmerl and David Garlan. Acmestudio : Supporting style-centered architecture development. In *Proceedings of the 26th International Conference on Software Engineering*, pages 704–705. IEEE Computer Society, 2004.

[12] Richard Mark Soley and Bill Curtis. The consortium for it software quality (cisq). In *International Conference on Software Quality*, pages 3–9. Springer, 2013.