DEQUALITE : méthode de construction de modèles de qualité prenant en compte la conception des systèmes

Foutse Khomh — Naouel Moha — Yann-Gaël Guéhéneuc

Équipe PTIDEJ– *GEODES*
Département d'informatique et de recherche opérationnelle, Université de Montréal
CP 6128 succ. Centre Ville, Montréal, Québec, H3C 3J7, Canada
{foutsekh, mohanaou, guehene}@iro.umontreal.ca

RÉSUMÉ. *La plupart des modèles de qualité présentés dans la littérature ou utilisés dans l'industrie pour évaluer les systèmes par objets utilisent des métriques de classes (nombre de méthodes d'une classe par exemple) ou des métriques de relations entre classes (couplage entre deux classes par exemple) pour mesurer les attributs internes des systèmes. Cependant, la qualité des systèmes par objets ne dépend pas uniquement de la structure de leurs classes mais aussi de la façon dont celles-ci sont organisées, c'est-à-dire de leur conception. Nous proposons DEQUALITE, une méthode de construction de modèles de qualité permettant de mesurer la qualité des systèmes par objets en prenant en compte non seulement les attributs internes du système mais aussi sa conception. Notre méthode utilise une approche par apprentissage. Elle s'appuie sur une étude des patrons de conception pour prendre en compte la conception des systèmes. Notre méthode permet aussi de combiner des modèles de qualité afin d'augmenter la capacité de prédiction. Nous illustrons notre méthode sur un ensemble de systèmes implantant des patrons de conception et sur le modèle de qualité QMOOD de Bansiya. Nous discutons les avantages et les inconvénients de cette méthode et procédons à la validation d'un modèle de qualité résultant sur un ensemble de systèmes. Nous terminons par une discussion sur les avantages et limitations de l'utilisation des patrons de conception pour la construction de modèles de qualité.*

ABSTRACT. *Object-oriented software quality models usually use metrics of classes (such as number of methods) or of relationships between classes (for example coupling) to measure internal attributes of systems. However, the quality of object-oriented systems does not depend on classes solely: It also depends on the organisation of classes,* i.e.*, the system design. We propose DEQUALITE, a method to build quality models that allows the measure of the quality of object-oriented systems taking into account*

Technique et science informatiques, pages 1 à 1

the internal attributes of the system but also its design. Our method uses a machine learning approach and also enables the combination of models for the improvement of the performance. We justify the use of patterns to build quality models, we illustrate our method on a set of systems implementing design patterns and on the quality model QMOOD from Bansiya. We discuss the advantages and limitations of this method, we then present a validation of a resulting quality model on a set of systems. We conclude on the advantages of using patterns to build software quality models and the difficulty of doing so.

MOTS-CLÉS : *patrons de conception, modèles de qualité, apprentissage*

KEYWORDS: *design patterns, quality models, machine learning*

1. Introduction

Dans cet article, nous présentons une méthode, DEQUALITE (Design Enhanced QUALITy Evaluation), pour la construction de modèles de qualité permettant de mesurer la qualité des systèmes par objets en prenant en compte non seulement les attributs internes des systèmes mais aussi leur conception. Dans le reste de cet article, nous utilisons le terme “conception” pour parler du *design* d’un système par objets et distinguons conception et architecture, cette dernière englobant une plus large portion d’un système (Eden *et al.*, 2003).

Il est reconnu que les systèmes informatiques sont souvent trop coûteux. Cette situation est due à plusieurs facteurs, parmi lesquels les plus importants sont : (1) l’imprécision des spécifications des utilisateurs car des spécifications imprécises provoquent des problèmes de communications et de planification lors du développement et de la maintenance qui gênèrent des pertes de temps qui augmentent les coûts et (2) la mauvaise qualité des systèmes (Bennett *et al.*, 2000) du point de vue des développeurs (par exemple, mauvaise maintenabilité) et des usagers (par exemple, faibles performances). Parmi tous ces facteurs, le seul sur lequel nous ayons un réel contrôle est la qualité des systèmes. Dans cet article, nous nous intéressons seulement à la qualité du point de vue des développeurs dans le contexte de la maintenance. En effet, ces dix dernières années, le coût de la maintenance des systèmes orientés objets s’est accru jusqu’à compter pour plus de 70% du coût total des systèmes (Pressman, 2001). Cependant, avant de pouvoir améliorer la qualité pour réduire le coût de la maintenance, il faut pouvoir évaluer cette qualité précisément.

De nombreux modèles de qualité ont été proposés dans la littérature faisant le lien entre attributs internes des systèmes et attributs externes de qualité afin d’évaluer la qualité des systèmes. Ces modèles utilisent des métriques pour mesurer les attributs internes des systèmes car les métriques sont désormais reconnues comme appropriées pour créer de tels modèles (Bansiya *et al.*, 2002). Les modèles existants se concentrent sur l’évaluation des attributs internes des classes (tels que la taille, la filiation, la cohésion) et entre classes (tels que le couplage) sans prendre en compte l’organisation de celles-ci. Ils font ainsi difficilement la différence entre un système bien conçu et un système ayant une conception médiocre.

Nous pensons que les attributs internes des classes et les dépendances entre les classes ne sont donc pas suffisants pour l’évaluation de la qualité des systèmes. En effet, l’organisation des classes et plus généralement la conception du système a un impact important sur les efforts de maintenance car ce sont les premières caractéristiques des systèmes auxquelles les développeurs sont confrontés.

De nombreux principes et techniques existent pour aider au développement de systèmes de bonne qualité ; parmi celles-ci, les patrons de conception font un pont intéressant entre les attributs internes des systèmes (leur implantation concrète) et leurs attributs externes tels que la réutilisabilité, la flexibilité et la modularité (Gamma *et al.*, 1994).

Depuis leur introduction par Gamma *et al.* (1994), les patrons de conception ont pris une place de plus en plus importante dans la construction des systèmes. Des auteurs comme Venners (2005) montrent que les patrons de conception ont un impact positif sur la qualité des systèmes. D'autres par contre, comme Wendorff (2001), suggèrent que leur utilisation dans certaines situations n'améliore pas toujours la qualité des systèmes. McNatt *et al.* (2001), quant à eux, montrent qu'une implémentation très couplée de patrons de conception, c'est-à-dire une implémentation dans laquelle on retrouve des classes jouant des rôles différents dans des patrons différents dégrade sensiblement la qualité globale d'un système.

Malgré ces différentes études qualitatives, de tous les modèles de qualité présentés dans la littérature, aucun ne prend en compte de façon explicite la conception du système. Ce manque dans les modèles de qualité est en partie dû au fait qu'aucune étude quantitative n'a été menée afin de mesurer l'impact des patrons de conception sur la qualité des systèmes.

La méthode DEQUALITE présentée ici s'appuie sur une étude de l'impact des patrons de conception sur la qualité des systèmes. Elle utilise des algorithmes d'apprentissage pour faire le lien entre les valeurs des attributs internes des systèmes et leurs attributs externes de qualité. Elle fait aussi usage d'une technique de combinaison d'expertises pour permettre la combinaison de modèles.

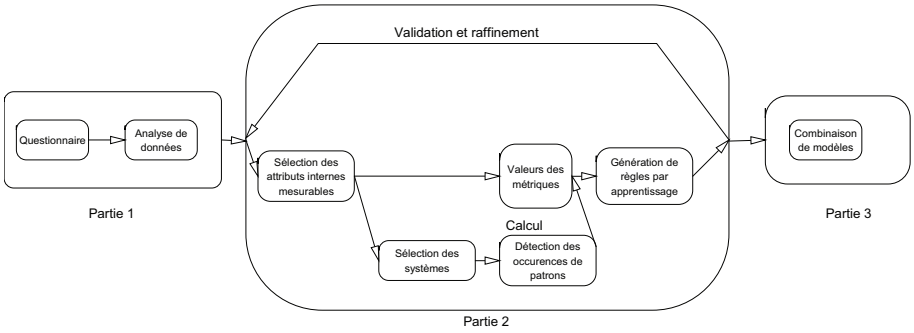

Figure 1. *Étapes de la méthode DEQUALITE.*

La contribution majeure de cet article est la méthode DEQUALITE, illustrée par la figure 1.

Suivant cette méthode, trois autres contributions sont présentées dans cet article :

– dans la partie 1, nous montrons que les patrons de conception ont un impact mesurable sur la qualité des systèmes (ce qui justifie la construction de modèles de qualité à partir de ceux-ci). Nous produisons une évaluation quantitative de l'impact des 23 patrons de conception de Gamma *et al.* (1994) sur dix attributs de qualité liés à la maintenance. Ceci est détaillé dans la section 2 de l'article ;

– dans la partie 2, nous présentons un modèle de qualité appelé PQMOD et constitué d'un ensemble de règles permettant l'évaluation de la qualité des systèmes en

prenant en compte leur conception. Nous discutons par la suite de le pertinence de ses règles vis-à-vis des bonnes pratiques de la programmation par objets. De plus, nous illustrons l'application de PQMOD sur un ensemble de systèmes implantant des patrons de conception. Ceci est détaillé dans la section 3 de l'article ;

– dans la partie 3, nous reprenons une technique permettant de fusionner PQMOD avec des modèles de qualité quelconques tirés de la littérature à base de règles ou d'arbres de décisions afin d'enrichir ceux-ci de l'expertise des patrons de conception contenue dans le modèle PQMOD. Nous illustrons cette fusion avec le modèle de qualité QMOOD de Bansiya *et al.* (2002). Ceci est détaillé dans la section 4 de l'article ;

Nous avons choisi pour la construction de notre méthode de suivre l'approche proposée par Dromey (1996) qui consiste à :

1) définir des attributs de qualité mesurables de haut niveau (attributs externes) ;

2) identifier et classifier des propriétés mesurables du système (attributs internes) ;

3) proposer un ensemble d'axiomes faisant le lien entre ces propriétés du système et les attributs de qualité de haut niveau ;

4) procéder à l'évaluation de la capacité de prédiction du modèle obtenu et éventuellement à son raffinement.

La section 5 présente une étude des travaux connexes et une discussion sur les avantages et les inconvénients de notre approche. La section 6 conclut cet article et introduit des travaux futurs.

2. Partie 1 : impact des patrons de conception sur la qualité des systèmes

L'utilisation de patrons de conception dans les systèmes peut être vu comme un style de programmation au sens artistique. En effet, si nous faisons le parallèle avec le domaine de l'art et regardons un système comme un tableau, les patrons de conception peuvent être vu comme le style de ce tableau et la qualité de ce style a un impact sur la qualité du tableau, peu importe ce qu'il montre. Par exemple, pour comparer une peinture cubiste, telle que "Femme Profile" de Pablo Picasso (1939), avec une photo réaliste (cf. figure 2), le style de chacune de ces peintures doit être pris en compte car les deux visages possèdent chacun deux yeux, un nez, deux oreilles et une bouche. Seule l'organisation des constituants des peintures est différente.

Nous pensons que la qualité d'un système dépend pareillement de sa conception ; l'organisation des ses constituants qui est souvent déterminée par des patrons de conception. Le lien entre peinture et logiciel est explicité dans le tableau 1 : la peinture est comparée au système et les yeux, le nez, les oreilles et la bouche aux constituants du système. L'organisation de la peinture est comparée à la conception du système, son style est comparé aux patrons de conception et les critiques d'art aux développeurs.

Figure 2. *Femme de profile : versions réaliste, cubiste, et ses constituants.*

Art	Programmation
Artistes	Développeurs, mainteneurs
Critiques d'art	Ingénieurs qualité, mainteneurs
Peinture	Système
Organisation	Conception et/ou architecture
Style	Patrons
Yeux, nez, oreilles, bouche	Classes, interfaces, méthodes, etc.

Tableau 1. *Parallèle entre peinture et conception des systèmes.*

Si nous voulons pouvoir prendre en compte les patrons de conception dans l'évaluation de la qualité d'un système, nous devons d'abord mesurer l'impact de ces patrons de conception sur la qualité des systèmes. L'étude de l'impact des patrons de conception sur la qualité des systèmes est comparable à l'étude du style d'un peintre sur ses tableaux.

Plusieurs études qualitatives suggèrent que les patrons de conception ont un impact sur la qualité des systèmes mais aucune d'entre elles ne l'a prouvé. Or, pour prendre en compte la présence de ces patrons de conception dans les systèmes lors de l'évaluation de la qualité, il est nécessaire de quantifier l'impact de ceux-ci sur la qualité des systèmes. C'est pourquoi dans la première partie de notre méthode DEQUALITE, nous allons prouver quantitativement que les patrons de conception ont un impact sur la qualité des systèmes.

Tout comme en peinture, nous avions le choix entre une étude absolue, une étude relative ou une étude empirique de l'impact des patrons de conception sur la qualité des systèmes. Une étude absolue consiste à évaluer théoriquement l'impact des patrons sur la qualité des systèmes suivant la méthode présentée. Une telle étude a l'avantage de s'appuyer sur des bases théoriques solides. Mais, les patrons de conception étant des solutions abstraites, tout comme les styles en peinture (où il y a plusieurs façons d'exécuter chaque style), il existe plusieurs implantations possibles pour chaque patron. Il est donc difficile d'évaluer l'impact de ceux-ci sur la qualité des systèmes en ne s'appuyant que sur des bases théoriques.

Une étude relative consiste par exemple à analyser l'impact des patrons de conception sur la qualité en utilisant des techniques d'apprentissage machine telles que présentées dans (Guéhéneuc *et al.*, 2004). Une telle étude aurait l'avantage d'évaluer l'impact des patrons de conception sur la base de leurs implantations concrètes. Pour poursuivre le parallèle avec la peinture, cela reviendrait à comparer les mêmes styles sur plusieurs tableaux. Cependant, cette étude serait subjective et donc difficilement généralisable à cause du manque de données : en effet, il est difficile d'identifier les patrons de conception dans les systèmes (Guéhéneuc *et al.*, 2008) et quasiment impossible d'identifier toutes les implantations possibles d'un patron.

Une étude empirique quant à elle consiste à collectionner des évaluations de l'impact des patrons de conception sur la qualité des systèmes par des ingénieurs. Une telle étude est intéressante car le développement logiciel et la maintenance sont avant tout des activités manuelles réalisées par des ingénieurs. Leurs évaluations sont donc importantes. Les résultats ainsi obtenus seront donc représentatifs de l'expérience des ingénieurs sur plusieurs implantations de patrons. Avec un nombre grandissant de répondants, une telle étude produira donc une évaluation plus précise de l'impact des patrons de conception sur la qualité des systèmes qu'une approche absolue ou relative. En poursuivant notre parallèle avec la peinture, une telle étude équivaut à une évaluation du style favori d'un ensemble de critiques d'art. Cette approche est plus adéquate qu'une étude absolue car elle s'appuie sur des implantations concrètes des patrons. Elle ne possède pas les limitations d'une étude relative car elle prend en compte toutes les variations possibles d'un patron.

Pour toutes ces raisons, nous avons entrepris une étude empirique afin de mesurer l'impact des patrons de conception sur des attributs de qualité liés à la maintenance. Nous avons choisi ici une approche par sondage auprès d'ingénieurs logiciels car seule une évaluation de l'impact des patrons de conception sur la qualité par les ingénieurs logiciels est capable de traduire l'impact des patrons sur la qualité des systèmes tel que perçu lors des activités de développement et de maintenance.

Dans la suite de cette section, nous présentons la première partie de notre méthode DEQUALITE qui consiste à montrer quantitativement que les patrons de conception ont un impact sur la qualité des systèmes. Cette première partie se subdivise en deux étapes suivantes :

2.1. *Étape 1 : définition du questionnaire*

Nous avons décomposé en sous-caractéristiques les attributs de qualité présentés dans IEEE, ISO/IEC 9126 et les modèles de McCall, Boehm et Firesmith (McCall, 2001; Boehm *et al.*, 1976; Firesmith, 2004). Pour la maintenabilité par exemple, les sous-caractéristiques sont : la facilité d'apprentissage, facilité de test, l'extensibilité et la stabilité. Parmi toutes ces sous-caractéristiques, nous en avons retenus dix pour lesquels nous pensons que les patrons de conception pourraient avoir un impact en

théorie. (Ce choix est confirmé par la suite sauf dans le cas particulier de Facilité d'apprentissage.) Voici la liste et leurs descriptions :

- attributs liés à la conception :
  - **extensibilité :** degré avec lequel l'architecture et les fonctionnalités peuvent être étendues ;
  - **simplicité :** degré de compréhension de l'architecture d'un système ;
  - **réutilisabilité :** degré de réutilisation des morceaux d'architecture ;
- attributs liés à l'implémentation :
  - **facilité d'apprentissage :** degré de facilité d'apprentissage du code pour de nouveaux développeurs ;
  - **compréhensibilité :** degré de facilité de compréhension du code source pour tous les développeurs, y compris ceux ayant participés au development original ;
  - **modularité :** degré d'indépendance entre les différentes fonctions ;
- attributs liés à l'exécution :
  - **généralité :** aptitude à exécuter un grand nombre de fonctions ;
  - **modularité à l'exécution :** degré d'indépendance entre les fonctions à l'exécution ;
  - **passage à l'échelle :** aptitude à s'exécuter correctement sur un très grand ensemble de données ;
  - **robustesse :** aptitude à fonctionner correctement même dans des conditions anormales ;

Chaque attribut de qualité a été évalué sur une échelle de Likert à six points :

```
A - Très Positif
B - Positif
C - Négligeable
D - Négatif
E - Très Négatif
F - Ne sait pas
```

Le sixième point permettait aux répondants de ne pas fournir de réponse s'ils n'étaient pas sûrs ou s'ils ignoraient l'impact du patron concerné sur l'attribut.

Pour chacun des 23 patrons de conception de Gamma *et al.* (1994) et pour chaque attribut de qualité listé ci-dessus, il était demandé à chacun des répondants d'évaluer l'impact du patron de conception sur la qualité d'un système dans lequel le patron aurait été utilisé de manière adéquate. Le questionnaire est disponible à l'adresse `http://www.ptidej.net/downloads/`.

Ce questionnaire a été distribué et publié sur trois listes de diffusions *refactoring*, *patterns-discussion* et *gang-of-4-patterns*, et les données présentées ci-après ont été collectées pendant la période de janvier à avril 2007.

2.2. *Étape 2 : analyse des données*

Parmi les 30 questionnaires reçus, nous en avons sélectionné 20 provenant d'ingénieurs ayant une grande expérience dans l'utilisation des patrons de conception : 18 étaient remplis complètement et 2 étaient remplis à 90 %. Ces 20 questionnaires et 5 autres parmi ceux que nous n'avons pas retenus provenaient d'ingénieurs jugeant leur expérience comme très bonne. Les 10 questionnaires que nous n'avons pas retenus étaient partiellement remplies et contenaient la mention `F` (Ne sait pas) pour beaucoup de questions.

Après analyse des données des questionnaires, il s'est avéré que la difference entre les options `Positif` et `Très Positif` n'était pas pertinente mais simplement le résultat d'une différence de sévérité parmi les différents répondants (respectivement Négatif, Très Négatif). Par exemple, pour le patron Builder et l'extensibilité, nous avons obtenu 19% de répondants considérant l'impact du patron `Très Positif` tandis que 63% le considéraient `Positif` et 18% `Neutre`. Nous avons donc décidé de fusionner les réponses `A` et `B` et les réponses `D` et `E` pour obtenir une évaluation conservatrice reflétant mieux l'impact des patrons de conception car cette fusion permet de limiter le biais dû à la difference de sévérité entre les répondants. Nous avons obtenu l'échelle de Likert à trois points suivante :

`Positif` = `A` et `B`
`Neutre` = `C`
`Négatif` = `D` et `E`

Les réponses `F` n'ont pas été considérées car elles représentent des situations où le répondant ne connait pas ou ne souhaite pas évaluer l'impact du patron.

Nous avons ensuite calculé avec cette échelle de Likert à trois points et pour chaque attribut de qualité, les fréquences des réponses : `Positif`, `Neutre` et `Négatif`. Les tableaux 5 et 6 présentent les évaluations par les répondants de l'impact des 23 patrons de conception de Gamma *et al.* (1994) sur les attributs de qualité. L'analyse de ces évaluations nous a révélé des résultats parfois surprenants. Par exemple, pour trois patrons de conception parmi les plus utilisés comme l'Objet composite, la Fabrique Abstraite et le Poids-mouche. Nous avons obtenu les résultats présentés ci-dessous.

2.2.1. *Analyse de résultats pour 3 patrons de conception représentatifs*

L'Objet composite. Il est perçu de manière globale comme améliorant la qualité des systèmes. Les résultats montrent que tous les attributs de qualité sont positivement impactés par celui-ci sauf le passage à l'échelle et la robustesse. Considérant la définition de l'Objet composite, un impact neutre sur le passage à l'échelle du système est surprenant.

La Fabrique abstraite. Elle est considérée comme améliorant cinq des dix attributs de qualité étudiés. Ce résultat est cependant un peu surprenant vu la complexité de ce patron. Chose encore plus frappante, le degré de facilité d'apprentissage et la com-

préhensibilité du système sont jugés négativement impactés par ce patron pourtant un but de celui-ci (Gamma *et al.*, 1994) est justement a priori de contribuer à une gestion claire de la création d'objets dans le système.

Le Poids-mouche. Il est, quant à lui, perçu comme ayant un impact négatif sur tous les attributs de qualité sauf le passage à l'échelle. Considérant le problème résolu par ce patron, son impact positif sur la mise à l'échelle du système est normal. De façon plus globale, la perception négative de ce patron pourrait être le résultat de la faible utilisation de celui-ci comparativement à l'Objet composite ou la Fabrique abstraite.

En s'attardant maintenant sur les trois principaux attributs promus dans le livre de Gamma *et al.* (1994), à savoir l'extensibilité, la réutilisabilité et la compréhensibilité, nous observons des résultats en contradiction avec ce qu'annonce ce livre.

2.2.2. *Analyse de résultats pour quelques attributs de qualité*

L'extensibilité. Il apparaît des évaluations par les répondants de l'impact des patrons de conception sur l'extensibilité du système que tous les répondants s'accordent sur le fait que les patrons de conception améliorent l'extensibilité des systèmes. Des 23 patrons, 19 sont considérés comme ayant un impact positif, 3 un impact neutre et 1 un impact négatif. Ce qui est conforme aux objectifs des patrons décrits dans (Gamma *et al.*, 1994).

La réutilisabilité. Elle est, quant à elle, jugée légèrement négativement impactée par les patrons de conception. Des 23 patrons, seulement 10 sont considérés comme ayant un impact positif, 6 ont un impact neutre et 7 un impact négatif. Ce résultat est d'autant plus surprenant que les patrons de conception sont présentés théoriquement comme améliorant la réutilisation des systèmes. Certes dans (Gamma *et al.*, 1994), il s'agit plus de la réutilisation de bonne solution mais qui est très souvent traduite en promesse de réutilisation du code de ces solutions.

La compréhensibilité. Les patrons de conception semblent impacter légèrement négativement la compréhensibilité des systèmes. Des 23 patrons, seulement 15 sont considérés comme ayant un impact positif, 3 ont un impact neutre et 5 un impact négatif. Ce qui est en opposition avec les objectifs des patrons décrits dans (Gamma *et al.*, 1994).

2.2.3. *Discussion et évaluation statistique*

Des observations précédentes, nous pouvons déjà remarquer que les patrons de conception ont un impact sur la qualité des systèmes et que, contrairement à ce qui est annoncé dans la littérature (Gamma *et al.*, 1994), plusieurs patrons de conception ont un impact neutre et parfois même négatif sur certains attributs de qualité des systèmes.

Afin de vérifier statistiquement l'impact des patrons sur la qualité, nous faisons l'hypothèse suivante : *les patrons de conception améliorent la qualité des systèmes*. Nous effectuons ensuite un test d'hypothèse nulle afin de confirmer ou réfuter celle-ci.

Sur la base des fréquences des réponses positives (`Positif`) et non-positives (nous avons combiné les réponses `Neutre` et `Négatif` car notre but ici est de vérifier si les patrons ont un impact positif), nous décidons de l'impact de chaque patron de conception sur chaque attribut de qualité.

Nous avons associé à chaque question de notre questionnaire une variable aléatoire X prenant la valeur 0 lorsque l'impact du patron sur l'attribut de qualité est `Positif` et 1 dans le cas contraire. Si P est la probabilité que le patron n'ait pas un impact positif sur l'attribut de qualité, la probabilité de l'événement contraire (le patron a un impact positif sur l'attribut de qualité) est donc $1-P$. Pour N répondants $j = 1, ..., N$, nous considérons leurs réponses comme des occurrences de la variable aléatoire X que nous notons : $X_1, X_2, ...X_N$.

Notre hypothèse nulle s'exprime alors comme $H_0 : P \leq \frac{1}{2}$; elle signifie que l'impact du patron de conception sur l'attribut de qualité est positif. L'hypothèse alternative est donc $H_1 : P > \frac{1}{2}$, qui signifie que le patron n'a pas un impact positif sur l'attribut de qualité.

Notre règle de décision est la suivante :

- nous confirmons H_0 si f_N n'est pas suffisamment grand ;
- nous confirmons H_1 si f_N est suffisamment grand ;

où f_N est la fréquence des répondants ayant répondu que le patron a un impact négatif ou neutre sur l'attribut de qualité. Par "suffisamment grand" nous faisons référence à un seuil ayant un impact direct sur le risque encouru par une décision à ce niveau. Par exemple, si suffisamment grand signifie $\geq 80\%$, le risque encouru par une décision à ce niveau est de $0,37\%$, par contre si suffisamment grand signifie $\geq 60\%$, ce risque est de $15,09\%$. Ces valeurs sont calculées grâce à la distribution de Bernoulli choisie en raison de la nature de la variable.

Le risque encouru par un rejet de l'hypothèse nulle H_0, c'est-à-dire, l'impact du patron de conception sur l'attribut de qualité est positif, est donc : $1 - F(f_N)$, où F est la densité de probabilité de la distribution de Bernoulli $\beta(N, \frac{1}{2})$.

Les résultats de ce test d'hypothèse nulle sont résumés dans les tableaux 7 et 8. Dans ces tableaux, le signe $+$ signifie qu'à l'issue de notre test d'hypothèse nulle, l'impact du patron sur l'attribut de qualité est positif. Le signe $-$ signifie le contraire (c'est-à-dire négatif ou neutre). Le nombre à côté du signe représente le risque encouru par la prise de cette décision.

En conclusion, nous pouvons affirmer que les patrons de conception ont bien un impact quantifiable et de manière générale positif sur la qualité des systèmes. Nous justifions ainsi l'intérêt de notre méthode DEQUALITE qui a pour objectif de construire des modèles de qualité prenant en compte la conception des systèmes à travers leurs implantations de patrons de conception.

Après avoir prouvé que les patrons de conception ont un impact sur la qualité des systèmes, nous avons ensuite mesuré cet impact. Le tableau 2 résume l'impact des 23 patrons de conception de Gamma *et al.* (1994) sur les dix attributs de qualité sélectionnés. Dans la section suivante, nous présentons la deuxième partie de notre méthode DEQUALITE qui consiste à construire un modèle de qualité prenant en compte les patrons de conception utilisés pendant la conception des systèmes.

	Caractéristiques de qualité									
Patrons de conception	Extensibilité	Simplicité	Généralité	Modularité	Modularité à l'exécution	Facilité d'apprentissage	Compréhensibilité	Réutilisabilité	Passage à l'échelle	Robustesse
Fabrique abstraite	+	+	+	+	+	−	+	+	=	=
Monteur	+	+	+	+	+	+	+	=	=	=
Fabrique	+	+	+	+	+	+	+	+	=	=
Prototype	+	+	+	=	+	+	+	+	=	=
Singleton	−	+	=	=	=	+	+	=	=	=
Adaptateur	+	+	+	+	+	+	+	+	−	=
Pont	+	−	+	+	+	−	+	−	−	=
Objet composite	+	+	+	+	+	+	+	+	+	=
Décorateur	+	=	+	+	+	+	−	−	−	=
Façade	+	+	+	+	−	+	+	=	=	=
Poids-mouche	=	−	−	−	=	−	−	−	+	=
Proxy	=	+	+	+	=	+	=	+	−	=
Chaine de responsabilité	+	+	+	+	+	+	−	+	=	=
Commande	+	+	+	+	+	+	−	−	−	=
Interpréteur	+	+	+	+	+	+	+	+	−	=
Itérateur	+	+	+	+	+	+	+	+	−	=
Médiateur	+	−	+	+	=	+	+	=	=	=
Memento	=	+	=	=	=	+	=	+	−	=
Observateur	+	+	+	+	+	+	+	+	=	=
État	+	+	+	+	+	+	+	−	=	=
Stratégie	+	+	+	+	+	+	+	+	=	=
Patron de méthode	+	+	+	=	=	+	+	+	=	=
Visiteur	+	−	+	+	+	−	−	−	−	=

Tableau 2. *Évaluation de l'impact des patrons de conception sur la qualité d'un système (Positif : +, Neutre : = et Négatif : −).*

3. Modèles de qualité prenant en compte les patrons de conception

La sélection des attributs de qualité de haut niveau ayant déjà été faite à l'étape 1 de la première partie, nous allons maintenant sélectionner des attributs internes mesurables et des systèmes implantant des patrons de conception ; nous mesurons ensuite sur ces systèmes ces attributs internes sélectionnés. Grâce à l'évaluation de l'impact des patrons de conception sur la qualité des systèmes obtenu à l'étape 2 de la première partie, nous procédons alors avec les algorithmes d'apprentissage JRip et J48 à la génération d'un ensemble de règles faisant le lien entre les attributs internes et les

attributs externes en prenant en compte la présence de patrons de conception. Nous procéderons ensuite à une validation et un raffinement du modèle obtenu.

Le résultat de cette Partie 2 est un modèle de qualité, PQMOD, à base de règles et d'arbres de décision. Chaque règle (ou arbre de décision) correspond à un attribut externe de qualité et permet d'associer une valeur pour cet attribut à chaque classe d'un système. Les règles sont construites en analysant uniquement les classes jouant un rôle dans un patron de conception car la partie 1 nous fournit une évaluation de l'impact des patrons sur la qualité. PQMOD traite ensuite chaque classe d'un système en faisant l'hypothèse qu'elle pourrait jouer un rôle dans un patron de conception : pour chaque classe du système, un ensemble de valeurs de métriques est calculé et utilisé par PQMOD pour décider par évaluation successive des conditions de chaque règle de la valeur de l'attribut de qualité à affecter à la classe. Si aucune condition n'est satisfaite par la classe, la valeur par défaut Neutre (=) lui est donnée. (La valeur par défaut peut-être remplacée par une valeur attribuée par un autre modèle de qualité, comme discuté dans la partie 3.) La qualité globale du système est obtenue en attribuant à chaque attribut externe de qualité la valeur la plus fréquente parmi ses classes.

Le modèle étant construit à partir de classes jouant des roles dans des patrons de conception, les résultats de PQMOD sont fiables uniquement pour des programmes incluant des patrons de conception ou dont les classes sont similaires à des classes jouant des roles dans des patrons de conception. Ainsi, comme tout modèle de qualité, ce modèle est approprié pour un contexte précis, à savoir celui des programmes implantant des patrons de conception.

Cette deuxième partie de la méthode DEQUALITE se résume en 5 principales étapes dont 3 sont constituées de préliminaires à la construction du modèle. Les 2 autres concernent la construction du modèle et sa validation. La figure 1 l'illustre.

3.1. *Choix préliminaires*

Étape 1 : sélection des attributs internes mesurables. Cette étape consiste à choisir des attributs internes pouvant être mesurés. Ces attributs seront reliés aux attributs externes sélectionnés à l'étape 1 de la première partie par apprentissage et automatiquement pour faire le lien entre constituants des systèmes et qualité à l'étape 4. Aussi, à cette étape, nous choisissons d'abord 29 métriques parmi les plus utilisées dans la littérature pour mesurer sur nos systèmes plusieurs propriétés concrètes qui sont entre autres : le couplage, la cohésion, la profondeur de l'arbre d'héritage, la connectivité, le nombre de méthodes, le nombre de champs, etc. ; pour plus de détails sur ces métriques, voir (Chidamber *et al.*, 1994), Briand *et al.* (1997), Hitz *et al.* (1995), Lorenz *et al.* (1994) et Tegarden *et al.* (1995). Nous utilisons indistinctement ces 29 métriques *a priori* pour la construction des règles dans les étapes suivantes car, à cette étape-ci, nous ne pouvons pas savoir quelles métriques seront pertinentes pour quels attributs externes.

Systèmes	Liste des patrons implantés	Nombre d'occurrences
QuickUML 2001	Fabrique abstraite, Monteur, Commande, Objet composite, Observateur, Singleton	7
Lexi v0.1.1	Monteur, Observateur, Singleton	5
JRefactory v2.6.24	Adaptateur, Monteur, Fabrique, Singleton, État, Visiteur	26
Netbeans v1.0	Fabrique abstraite, Adaptateur, Commande, Itérateur, Observateur	28
JUnitv3.7	Objet composite, Décorateur, Itérateur, Observateur, Singleton	8
JHotDraw v5.1	Adaptateur, Commande, Objet composite, Décorateur, Fabrique, Observateur, Prototype, Singleton, État, Stratégie, Patron de méthode	24
MapperXML v1.9.7	Fabrique abstraite, Adaptateur, Objet composite, Facade, Fabrique, Observateur, Singleton, Stratégie, Patron de méthode	16
Nutch v0.4	Singleton, Pont , Commande, Memento, Patron de méthode, Adaptateur, Stratégie, Itérateur	16
PMD v1.8	Adaptateur, Monteur, Objet composite, Fabrique	8

Tableau 3. *Systèmes de notre $\mathcal{BS}$.*

Étape 2 : sélection de systèmes implantant des patrons de conception. Cette étape consiste à sélectionner un ensemble de systèmes contenant des patrons de conception. Cet ensemble est appelé base de systèmes, notée $\mathcal{BS}$. À cette étape, nous considérons les systèmes : QuickUML 2001, Lexi v0.1.1, JRefactory v2.6.24, Netbeans v1.0, JUnit v3.7, JHotDraw v5.1, MapperXML v1.9.7, Nutch v0.4, PMD v1.8. Le tableau 3 présente la liste des patrons implantés dans ces systèmes ainsi que leur nombre d'occurrences. Dans ce cas précis et pour plus de précision, l'identification des patrons a été réalisée manuellement par des développeurs indépendants et ensuite compilée dans une base de données XML (Bieman *et al.*, 2003; Guéhéneuc *et al.*, 2004). Des techniques automatiques auraient aussi pu être utilisées mais pour cette première étude nous avons souhaité une bonne précision. En effet, les techniques de détection automatiques produisent encore pour la plupart un grand nombre de faux positifs et de duplications d'occurrences.

Étape 3 : mesure des attributs internes sur les patrons des systèmes de $\mathcal{BS}$. Cette étape consiste à calculer les valeurs des métriques sélectionnées précédemment sur les patrons identifiés dans les systèmes de $\mathcal{BS}$, c'est-à-dire calculer les métriques sur les classes de chacune des occurrences des patrons identifiés. Nous calculons ces métriques uniquement sur les classes des occurrences des patrons car nous cherchons un lien entre la structure de ces classes et leur impact sur les attributs externes de qualité

choisis à l'étape 2 de la première partie. Les métriques présentées ci-dessus et utilisées ici sont implantées dans POM (Guéhéneuc *et al.*, 2004).

3.2. *Étape 4 : construction de règles*

Cette étape consiste à appliquer une technique d'apprentissage pour faire le lien entre les attributs internes (valeurs des métriques de l'étape précédente) et les attributs externes (sélectionnés à l'étape 1 et évalués à l'étape 2 de la première partie). Nous avons appliqué ici les algorithmes d'apprentissage J48 et JRip pour inférer des règles à partir des données compilées dans les étapes 1 et 2 et nous avons sélectionné les règles ayant les meilleurs taux de classification. J48 est un équivalent de c4.5 (Quinlan, 1993) (algorithme pour la génération d'arbres de décision) développé sur la plateforme Weka (Witten *et al.*, 1999) et JRip est une implémentation sur la plateforme Weka de l'algorithme RIPPER (Cohen, 1995) qui est un classifieur par système de règles. Nous appelons PQMOD, le modèle de qualité constitué des dix règles suivantes :

Règle 1 : extensibilité.

```
NOA <= 2
|  AID <= 0,5 : Valeur : =
|  AID > 0,5 : Valeur : -
NOA > 2
| NOC <= 0
|  |  DIT <= 0,83 : Valeur : +
|  |  DIT > 0,83 : Valeur : -
| NOC > 0 : Valeur : +
```

Pour cette règle, nous avons obtenu un taux de classification de 89,36%. Selon cette règle, pour une bonne extensibilité, le nombre moyen d'ancêtres (NOA) doit être > 2 et chaque classe doit avoir au moins un fils ou bien une profondeur d'héritage (DIT) $\leq 0,83$. Cette règle est bien en accord avec ce que recommande les bonnes pratiques de la programmation par objets, comme le principe des hiérarchies abstraites et profondes (Martin, 2002).

Règle 2 : simplicité.

```
=> Simp : Valeur : +
```

Pour cette règle, nous avons obtenu un taux de classification de 90,42%. Elle souligne la simplicité des architectures implantant des patrons de conception, ce qui est bien en accord avec ce que revendique Gamma *et al.* (1994).

Règle 3 : généralité.

```
NOC <= 0
|  AID <= 0,83
|  |   AID <= 0,11 : Valeur : =
|  |   AID > 0,11 : Valeur : +
|   AID > 0,83 : Valeur : =
NOC > 0
|   ACMIC <= 1 : Valeur : +
|   ACMIC > 1 : Valeur : =
```

Nous avons obtenu pour cette règle un taux de classification de 90,50%. D'après cette règle, pour une bonne généralité (*i.e.*, le système peut exécuter facilement un grand nombre de fonctions), chaque classe doit avoir une profondeur d'héritage moyenne (AID) entre 0,11 et 0,83 ou bien être très faiblement couplée avec ses ancêtres (ACMIC). Cette règle est confirmée par les résultats expérimentaux de Harrisson *et al.* (Harrison *et al.*, 2000) qui indiquent qu'une profondeur d'héritage inférieure à trois facilite la généralité. Cette règle est bien en accord avec les bonnes pratiques de la programmation par objets qui recommande de coupler faiblement les classes entre elles et de faire des hiérarchies abstraites (Martin, 2002).

Règle 4 : modularité.

```
NCM <= 50,63
|  NOC <= 0
|  |   AID <= 0,83
|  |  |   AID <= 0,11 : Valeur : =
|  |  |   AID > 0,11 : Valeur : +
|  |   AID > 0,83 : Valeur : =
|   NOC> 0 : Valeur : +
NCM > 50,63
|   NOP <= 1,65 : Valeur : =
|   NOP> 1,65 : Valeur : +
```

Le taux de classification de cette règle a été de 79,79%. Selon elle, pour une bonne modularité, chaque classe doit avoir en moyenne pour un nombre de parents (NOP) > 1,65, un nombre de méthodes redéfinies (NCM) > 50,63. Dans le cas où elle a moins de parents, elle doit avoir au moins un fils ou une profondeur moyenne d'héritage (AID) > 0,11. Cette règle souligne la nécessité d'utiliser le polymorphisme d'héritage et le principe des hiérarchies abstraites et profondes, comme recommandé dans les bonnes pratiques de la programmation par objets (Martin, 2002).

Règle 5 : modularité à l'exécution.

```
(LCOM5 >= 1,02)
  et (DCAEC <= 0) => ModExec : Valeur : =
(NCM >= 62,78) => ModExec : Valeur : =
(NOA <= 2) => ModExec : Valeur : =
=> ModExec : Valeur : +
```

Le taux de classification de cette règle a été de 82,98%. D'après cette règle, pour une bonne modularité à l'exécution, il faut soit un nombre moyen de re-définitions de méthode (NCM) < 62,78, soit une bonne cohésion entre méthodes (LCOM5) < 1,02, soit au moins 3 ancêtres pour chaque classe. Cette règle nous donne une borne supérieure sur le nombre de re-définitions de méthodes pour une classe, soulignant ainsi qu'il faut éviter d'abuser du polymorphisme. De plus, elle recommande une bonne cohésion. Tout ceci est en accord avec ce que nous recommandent les bonnes pratiques de la programmation par objets (Martin, 2002).

Règle 6 : facilité d'apprentissage.

```
=> FacilAppr : Valeur : +
```

Avec un taux de classification de 99,42%, cette règle souligne la facilité d'apprentissage des architectures implantant des patrons de conception, ce qui est en accord avec les résultats de plusieurs travaux tels que (Lange *et al.*, 1995).

Règle 7 : compréhensibilité.

```
NCM <= 4
| CBO <= 1 : Valeur : =
| CBO > 1 : Valeur : +
NCM > 4
| ACMIC <= 0 : Valeur : +
| ACMIC > 0
|  | CLD <= 0,46
|  |  | NOD <= 1,63
|  |  |  | NOP <= 4,67
|  |  |  |  | DCMEC <= 0,2 : Valeur : +
|  |  |  |  | DCMEC > 0,2
|  |  |  |  | | AID <= 2,15 : Valeur : -
|  |  |  |  | |  AID > 2,15 : Valeur : +
|  |  |  | NOP> 4,67 : Valeur : -
|  |  | NOD > 1,63 : Valeur : -
|  | CLD > 0,46 : Valeur : +
```

Avec un taux de classification de 85,11%, cette règle stipule que pour une bonne compréhensibilité, il faut que les classes re-définissant plus de 4 méthodes aient un faible couplage avec leur ancêtres ou une "hauteur" (CLD) > 0,46 et un faible couplage avec leurs descendants. Cette règle confirme l'intuition que pour facilement comprendre une classe, il ne faut pas que celle-ci soit trop fortement couplée à d'autres classes. Elle est confirmée aussi par les travaux de Harrisson *et al.* (Harrison *et al.*, 2000). Elle souligne ainsi la nécessité de faire des hiérarchies profondes et abstraites avec un faible couplage, ce qui est en accord avec les bonnes pratiques de programmation et les travaux précédents de (Harrison *et al.*, 2000).

Règle 8 : réutilisabilité.

```
(NOC <= 0,23)
  et (AID >= 1) => Reus : Valeur : =
(NMA>= 16) => Reus : Valeur : =
=> Reus : Valeur : +
```

Le taux de classification de cette règle a été de 76,59%. Selon elle, toute classe ayant un nombre de fils (NOC) > 0,23 ou un nombre de nouvelles définitions de méthodes (NMA) < 16 est réutilisable. Elle confirme que si une classe à des fils c'est qu'elle déjà fertilisée partiellement par ses fils et que si une classe définit trop de nouvelles méthodes, elle devient complexe et difficile à réutiliser. Cette règle souligne la nécessité d'éviter le "mauvais" héritage qui consiste à mettre ensemble des objets structurellement différents mais ayant des méthodes semblables (héritage de fonctionnalités) et qui aboutit à une trop grande définition de nouvelles méthodes dans les sous-classes, ce qui s'accorde avec les résultats obtenus par Denier *et al.* (2008).

Règle 9 : passage à l'échelle.

```
(DCMEC >= 0,1)
   et (NOC <= 0,65)
   et (DCMEC <= 0,2) => Scal : Valeur : +
(AID >= 2,83)
   et (AID <= 2,92) => Scal : Valeur : +
(connectivity <= 0,66)
   et (NOC >= 0,75)
   et (ACMIC <= 0,2) => Scal : Valeur : -
(LCOM5 <= 0,63)
   et (ICHClass >= 0,25)
   et (NMA <= 5,88) => Scal : Valeur : -
=> Scal : Valeur : =
```

Le taux de classification de cette règle a été de 69,15%. Selon cette règle, pour un bon passage à l'échelle, il faut un faible nombre moyen d'enfants (NOC) < 0,65 et un nombre de descendants exportant le couplage (DCMEC) entre 0,1 et 0,2 ou bien une profondeur moyenne d'héritage (AID) entre 2,83 et 2,92. Cette règle est intéressante car elle décrit une structure similaire à la structure préconisée par le patron de conception Poids-mouche, qui, lorsqu'implanté, améliore le passage à l'échelle. Or, dans l'ensemble d'apprentissage $\mathcal{BS}$, ce patron n'est présent explicitement dans aucun des systèmes. Cette règle laisse donc penser que ce patron est utilisé implicitement et, possiblement, incomplètement pour tous les système ayant un bon passage à l'échelle. Elle souligne aussi l'importance d'un faible couplage, qui fait partie des bonnes pratiques de la programmation par objets (Martin, 2002).

Règle 10 : robustesse.

```
=> Robus : Valeur : =
```

Avec un taux de classification de 98,60%, cette règle souligne l'impact négligeable des patrons de conception sur la robustesse des systèmes, ce qui est en accord avec les évaluations des répondants de notre questionnaire.

3.3. *Étape 5 : validation du modèle de qualité PQMOD sur d'autres occurrences connues de patrons dans d'autres systèmes*

À cette étape, nous opérons une double validation, tout d'abord nous appliquons les règles du modèle de qualité PQMOD sur les occurrences des patrons de conception implantés dans le système PADL (Guéhéneuc, 2005) et comparons les résultats obtenus avec ceux du tableau 2 ; nous avons bien retrouvé pour les patrons Visiteur, Observateur, Fabrique Abstraite et Objet composite présents dans PADL la qualité annoncée dans le tableau 2. Ensuite, nous avons opéré un passage à l'échelle des différentes règles de PQMOD (la section suivante présente les détails de ce passage à l'échelle) et nous avons appliqué les règles obtenus de ce passage à l'échelle sur 5 systèmes ; nous comparons ensuite les résultats obtenus à une évaluation manuelle de la qualité de ces systèmes par un groupe d'experts indépendants dont certains ayant participé au développement de ces systèmes ; le tableau 4 présente les détails de ces résultats.

3.4. *Analyse du modèle de qualité PQMOD*

Dans la deuxième partie de notre méthode DEQUALITE, nous avons obtenu le modèle de qualité PQMOD constitué de l'ensemble des règles présentées à l'étape 4 et validées à l'étape 5. Ce modèle de qualité par sa construction permet une évaluation de la qualité des systèmes en prenant en compte leurs conceptions. Il comporte pour chaque caractéristique de qualité une règle ou un arbre de décision permettant d'évaluer celle-ci en fonction des valeurs de certaines métriques sur le système.

Avant d'appliquer ce modèle de qualité à un système quelconque $\mathcal{S}$, nous devons procéder d'abord à sa calibration. Nous utilisons pour cela une technique tirée de la métrologie.

Une calibration selon la norme de métrologie (ISO, 1993) est définie comme un ensemble d'opérations qui établissent, dans des conditions spécifiées, la relation entre les valeurs des quantités indiquées par un instrument de mesure (ou les valeurs représentées par une mesure matérielle) et les valeurs correspondantes réalisées par les normes. Le résultat de la calibration peut être considéré comme une correction des valeurs indiquées par l'instrument de mesure.

Dans notre cas, nous commençons par mesurer sur ce système les attributs internes identifiés précédemment. Par la suite, nous calculons des ratios entre les valeurs de métriques de $\mathcal{BS}$ et de celles de $\mathcal{S}$. En effet, les règles sont obtenues à partir de métriques ayant certaines valeurs minimales et maximales dépendantes de $\mathcal{BS}$. Ces valeurs sont différentes des minimales et maximales des valeurs des mêmes métriques calculées sur $\mathcal{S}$. Nous calculons donc le ratio entre $min_{\mathcal{BS}}$ et $max_{\mathcal{BS}}$, d'une part et $min_{\mathcal{S}}$ et $max_{\mathcal{S}}$, d'autre part, afin d'adapter leurs échelles, comme montré sur la figure 3. Les valeurs des métriques de $\mathcal{S}$ sont donc ramenées sur la même échelle que celles de $\mathcal{BS}$ et nous pouvons appliquer les règles du modèle PQMOD pour évaluer la qualité du système $\mathcal{S}$.

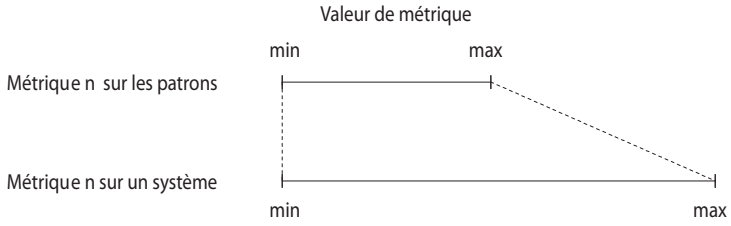

Figure 3. *Adaptation du modèle par ratio entre les minimum et maximum des valeurs des métriques pour $\mathcal{BS}$ et $\mathcal{S}$.*

3.4.1. *Analyse qualitative*

Comme nous l'avons mentionné à l'étape 4, ces règles sont bien en accord avec certaines bonnes pratiques que recommande le paradigme de la programmation par

Attributs	Nutch v0.4		Xerces v 1.4.4		Ant v1.7.0		GanttP. v2.0.4		PADL	
	Attendu	Prédit	Attendu	Prédit	Attendu	Prédit	Attendu	Prédit	Attendu	Prédit
Extensibilité	+	+	+	+	+	+	−	+	+	+
Généralité	=	+	+	+	+	+	−	+	=	+
Modularité	+	+	+	+	+	+	=	+	+	+
Mod. à l'exécution	+	+	=	+	+	=	−	=	=	+
Compréhensibilité	=	+	=	+	+	=	=	+	=	+
Réutilisabilité	+	+	=	+	+	+	−	+	=	+
Pass. à l'échelle	+	−	+	=	+	=	+	−	−	−

Tableau 4. *Résultats de l'application du modèle de qualité PQMOD sur quelques systèmes.*

objets. En plus de leurs liens avec les bonnes pratiques de programmation, ces règles nous fournissent des seuils permettant de mesurer la qualité des systèmes. Elles confirment donc les bonnes pratiques de programmation, sans jamais les contredire, tout en précisant des seuils pour des évaluations concrètes. Cependant, nous ne prétendons pas que les règles obtenues incluent toutes les conditions nécessaires et suffisante pour une évaluer les programmes en fonction des bonnes pratiques de la programmation. Ceci est aussi une raison de fusionner notre modèle avec d'autres modèles existants qui le complémenteraient.

3.4.2. *Analyse quantitative*

L'application du modèle de qualité PQMOD sur un ensemble de systèmes quelconques a donné les résultats du tableau 4. Dans ce tableau, des résultats de l'évaluation manuelle et ceux prédits par PQMOD sont présentés pour 5 systèmes.

Nous avons procédé parallèlement à une évaluation manuelle de la qualité de ces systèmes à travers un sondage auprès de plusieurs développeurs ayant utilisé et contribué au développement de ces systèmes. Comme attendu, des résultats obtenus pour ces 5 systèmes, seul le résultat de GanttProject s'est avéré être pour certains attributs de qualité en opposition avec la perception de plusieurs experts travaillant sur ce système. Ce resultat etait attendu car, comme souligné aussi par les experts, ce système ne contient pas de patrons de conception. Nous avons aussi appliqué notre modèle sur d'autres systèmes faisant un faible usage des patrons de conception comme ArgoUML et Azureus, cette tendance de prédiction a été confirmée.

Nous avons également analysé chacune des règles de PQMOD et appliqué ce modèle sur plusieurs versions de Xalan, Xerces et JFreechart ; nous avons observé une progression de la qualité de ces systèmes au fur et à mesure de leur évolution. Partant du principe que l'évolution des systèmes dans le temps se fait avec un souci d'amélioration de la qualité, nous pouvons donc dire que ce modèle permet bien de mesurer la qualité des systèmes.

En résumé, nous pouvons dire que PQMOD a une bonne capacité de prédiction sur les systèmes comportant un grand nombre de patrons de conception et reste limité pour des systèmes faisant peu usage de l'héritage (la plupart des règles font intervenir l'héritage) et n'implantant pas beaucoup de patrons de conception comme GanttProject ou, dans une moindre mesure, Xerces. C'est pourquoi, afin de bénéficier du pouvoir de prédiction de ce modèle sur les systèmes en général tout en s'assurant d'une prédiction au moins aussi bonne que celles des modèles présentés dans la littérature comme QMOOD de Bansiya *et al.* (2002), nous présentons dans la troisième partie de la méthode DEQUALITE une technique de combinaison de modèles qui permettra de combiner le modèle PQMOD avec des modèles présents dans la littérature.

4. Combinaison d'expertises sur l'évaluation de la qualité

Dans cette section, nous présentons la Partie 3 de la méthode DEQUALITE qui consiste à enrichir les modèles de qualité présents dans la littérature et qui sont à base de règles ou d'arbres de décisions de l'expertise du modèle PQMOD obtenu précédemment afin de leur permettre de prendre en compte la conception des systèmes dans leurs évaluations. La technique de combinaison d'expertises présentée est inspirée en partie des travaux de Bouktif (2005).

Avant de présenter cette technique de combinaison d'expertises, nous commençons par préciser quelques notations.

Notations. Soit T un arbre de décision donné, nous noterons $\mathcal{V}$ l'espace d'entrée de la fonction $f : \mathcal{V} \mapsto \mathcal{C}$ qui prédit la valeur $y_i \in \mathcal{C}$ d'un facteur de qualité y pour une observation $x_i \in \mathcal{V}$. $\mathcal{V}$ est défini par le produit cartésien $A_1 \times . . . \times A_d$, où A_j est le domaine de définition du $j^{ième}$ attribut et d est le nombre d'attributs d'entrée.
L'espace de sortie $\mathcal{C}$ est habituellement un ensemble ordonné $c_1, c_2, ..., c_q$ d'étiquettes ou de labels.

Chaque noeud interne de T est un test de type $x^{(j)} < \alpha_j$, avec $j = 1, ..., d$, $x^{(j)}$ un attribut de l'observation x et α_j une constante appartenant au domaine de définition de l'attribut $x^{(j)}$. Chaque attribut $x^{(j)}$ prend ses valeurs dans un domaine de définition limité par deux bornes L_j et U_j, respectivement, inférieure et supérieure.

Cet arbre T peut être représenté par un partitionnement binaire S d'un espace $\mathcal{V} \subset \mathbb{R}^d$ défini par le produit cartésien des intervalles $[L_j, U_j], j = 1, ..., d$: $\mathcal{V} = [L_1, U_1] \times \ldots \times [L_d, U_d]$.
Le partitionnement S revient à subdiviser récursivement l'espace $\mathcal{V}$ au moyen d'un hyperplan h_j d'équation $x^{(j)} = \alpha_j$ en deux sous-espaces. Un hyperplan h_j est associé à chaque noeud interne de test $x^{(j)} < \alpha_j$. La subdivision se poursuit jusqu'à ce que chaque sous-espace ne contienne que les points ayant la même classe. Le partitionnement S nous permet de voir un arbre de décision comme un ensemble d'hyperrectangles isothétiques (c'est-à-dire, ayant des faces parallèles aux axes des attributs). Ainsi, un modèle de type arbre de décision peut être codé par un vecteur de paires

(hyperrectangle, label). Un hyperrectangle (aussi appelé d-rectangle) R_k est défini par un produit cartésien d'intervalles $[l_j, u_j]$, $R_k = [l_1, u_1] \times \ldots \times [l_d, u_d]$, avec l_j et u_j étant les coordonnées des arrêts du d-rectangle, sur l'axe de l'attribut $x^{(j)}$.

Application. La combinaison d'expertises consiste à "unifier" l'espace d'entrée des arbres de décisions de tous les modèles candidats à la combinaison. Chaque arbre aura comme attributs d'entrée toutes les métriques de tous les arbres. Chaque métrique aura les mêmes bornes, respectivement, supérieure et inférieure dans tous les arbres. Ainsi, seul le partitionnement de l'espace d'entrée $\mathcal{V}$ distingue un arbre d'un autre.

Exemple. Pour illustrer cette opération, nous considérons deux arbres de décision qui prédisent le même facteur de qualité X. Le premier utilise les deux métriques NPPM et LCOMB et le deuxième utilise les deux métriques DIT et LCOMB. Après unification, les deux arbres auront les trois métriques NPPM, LCOMB et DIT comme attributs d'entrée et un espace d'entrée $\mathcal{V}$ défini par le produit cartésien : $\prod_{NPPM,LCOMB,DIT}[L_i, U_i]$. La figure 4 illustre cette unification.

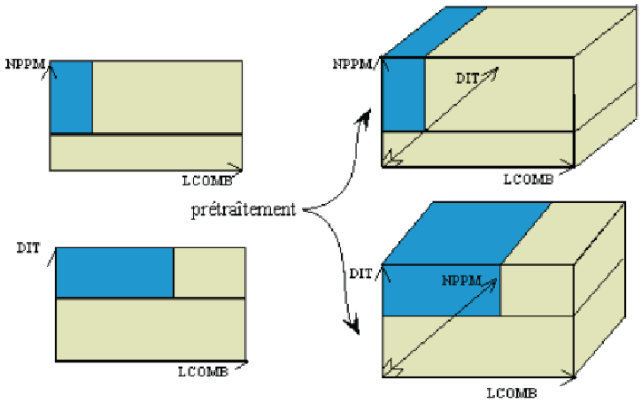

Figure 4. *Unification des espaces d'entrée des arbres de décision en vue de leur fusion (cette figure est tirée de (Bouktif, 2005))*

Après unification de ces espaces d'entrée, un arbre de décision unique est construit par fusion des ensembles de d-rectangles de tous les arbres à combiner. Ces arbres n'étant pas forcément indépendants. Plusieurs techniques sont proposées dans la littérature pour cette fusion. Entre autre nous pouvons citer les techniques de croisement intuitif, de croisement par greffe de d-rectangles et de croisement par superposition des couches de d-rectangles.

Dans le cadre de la méthode DEQUALITE, nous fusionnons les modèles de qualité avec le modèle PQMOD de la manière suivante : pour fusionner PQMOD avec un modèle quelconque Q, nous remplaçons les zones de faible prédiction de PQMOD,

c'est-à-dire les paires (hyperrectangle, label) (H, L) pour lesquelles la prédiction L est jugée mauvaise en terme de précision par (H, LQ), où LQ est la prédiction du modèle Q sur l'hyperrectangle H.

Pour exemple, nous appliquons cette technique sur le modèle de qualité QMOOD. Nous avons choisi QMOOD de Bansiya *et al.* (Bansiya *et al.*, 2002) parce qu'il est considéré dans la littérature comme l'un des modèles les plus aboutis et les mieux validés. De plus, nous disposons d'une implémentation de ce modèle. Nous obtenons après fusion des modèles PQMOD et QMOOD de (Bansiya *et al.*, 2002) le résultat suivant pour l'extensibilité :

```
NOA <= 2
|  AID <= 0,5 : Valeur : =
|  AID > 0,5 : Valeur : -
NOA > 2
| Valeur : (0,5)* ANA - (0,5)* DCC
+ (0,5)* MFA + (0,5)* NOPM
```

Cette règle de décision combine donc la force de prédiction du modèle PQMOD qui tient compte de l'architecture des systèmes ainsi que celle du modèle QMOOD qui a déjà été validée dans la littérature sur plusieurs systèmes n'utilisant pas particulièrement les patrons de conception. Ainsi, en combinant toutes les règles de PQMOD et QMOOD, nous obtenons un modèle de qualité prenant en compte la conception des systèmes pendant l'évaluation de leur qualité.

La méthode DEQUALITE présentée dans cet article permet donc, étant donné un modèle de qualité quelconque tiré de la littérature et qui est à base de règles ou d'arbres de décisions, de construire un modèle de qualité prenant en compte la conception des systèmes. La méthode DEQUALITE améliore ainsi l'évaluation de la qualité des systèmes par objets.

5. Travaux connexes

Nous présentons ici quelques travaux majeurs dans le domaine des modèles de qualité tout en soulignant le fait qu'aucun de ces travaux n'a proposé de modèles de qualité prenant en compte la conception des systèmes.

Briand *et al.* (2002) ont présenté une étude détaillée et exhaustive de la littérature sur les modèles de qualité. Ils classifient les modèles de qualité en deux catégories : les modèles basés sur des études de corrélation et ceux basés sur les expériences. La catégorie des modèles basés sur des études de corrélation sont issus d'analyses univariantes et multivariantes, tandis que ceux basés sur des expériences sont issus, par exemple, d'analyses de variance (ANOVA). Briand et Wüst argumentent qu'aucun de ces modèles de qualité ne mesure la qualité des systèmes en prenant en compte leur conception ; ils se contentent tous de mesurer la structure des classes.

Harrison *et al.* (2000) ont étudié la structure des systèmes orientés objets et ont uniquement fait le lien entre la modifiabilité, la compréhensibilité des systèmes et les

caractéristiques de l'héritage dans le système. En fait, ils ont réalisé une experience sur des systèmes ayant zéro, trois et cinq niveaux d'héritage. Ils ont observé que les systèmes sans héritage sont plus facile à modifier et à comprendre que ceux avec trois et cinq niveaux d'héritage. Cependant pour les grand systèmes, ils n'ont observés aucune influence du niveau d'heritage sur la compréhensibilité. Cette étude reste assez limitée car la modifiabilité et la compréhensibilité ne constituent qu'une partie des attributs de qualité d'un système.

Wydaeghe *et al.* (1998) ont évalué la qualité de l'architecture d'un éditeur OMT à travers l'étude de 7 patrons de conception. Plus précisément, ils ont sélectionnés 7 patrons de conception parmi les 23 présentés dans Gamma *et al.*(Gamma *et al.*, 1994) et les ont appliqués au développement d'un éditeur OMT. Dans leur article, ils discutent des difficultés liées à la selection et à l'application de ces patrons et terminent par leur évaluation subjective à partir de leur expérience personnelle de l'impact de ceux-ci sur la flexibilité, la modularité, la réutilisabilité et la compréhensibilité du système. Cependant, il n'ont fait aucune référence aux modèles de qualité dans leur étude.

Ainsi, bien que certaines études aient évaluées des caractéristiques architecturales des systèmes, aucune n'a proposé de modèles de prédiction de la qualité prenant en compte la conception des systèmes.

6. Conclusion

Dans cet article, nous avons proposé la méthode DEQUALITE de construction de modèles de qualité prenant en compte la conception des systèmes lors de leur évaluation. Cette méthode se décompose en trois parties et 8 étapes. Elle permet à partir de modèles de qualité quelconques tirés de la littérature et qui sont à base de règles ou d'arbres de décisions de construire des modèles de qualité prenant en compte la conception des systèmes. Nous avons validé les différentes étapes de construction de modèles de qualité de cette méthode sur plusieurs systèmes. L'utilisation des patrons de conception pour la construction d'un modèle de qualité résulte de notre choix d'étudier les caractéristiques de qualité architecturale plutôt que les caractéristiques quantitatives telles que la prépondérance des fautes. Les modèles de qualité construits à l'aide de notre méthode DEQUALITE évaluent la qualité des systèmes à un niveau d'abstraction plus élevé comparativement aux modèles existants car il utilise les patrons de conception comme base, plutôt que les classes.

L'utilisation des patrons de conception constitue une étape importante vers la construction de modèles de qualité capables d'évaluer les systèmes en prenant en compte non seulement leur conception mais aussi des informations architecturales plus détaillées, telles que la densité de patrons et/ou la présence d'anti-patrons (Brown *et al.*, 1998), par exemple.

Dans nos travaux futurs, nous enrichirons à l'aide de mesures d'architectures, comme suggérées par Kerievsky (2004), notre méthode en incluant la densité de pa-

trons, des mesures d'interactions entre ceux-ci et la présence de défauts de conception (tels que les anti-patrons (Brown *et al.*, 1998), et les mauvaises odeurs (Fowler, 1999)). Nous validerons également les modèles obtenus sur un plus large ensemble de systèmes. Nous étudierons la relation entre fautes et conception des systèmes. Nous évaluerons l'impact du nombre d'occurrences sur les modèles de qualité obtenus. Nous explorerons aussi l'impact des implantations "canonique" des patrons de conception hors de tout système sur la qualité.

Remerciements

Les auteurs sont partiellement soutenus par une subvention à la découverte du CRSNG.

7. Bibliographie

Bansiya J., Davis C. G., « A hierarchical model for object-oriented design quality assessment », *IEEE Transactions on Software Engineering*, vol. 28, p. 4-17, January, 2002.

Bennett K. H., Rajlich V., « Software Maintenance and Evolution : a Roadmap », *in* A. Finkelstein (ed.), *The Future of Software Engineering*, ACM Press, 2000.

Bieman J., Straw G., Wang H., Munger P. W., Alexander R. T., « Design Patterns and Change Proneness : An Examination of Five Evolving Systems », *in* M. Berry, W. Harrison (eds), *Proceedings of the* 9^{th} *international Software Metrics Symposium*, IEEE Computer Society Press, p. 40-49, September, 2003.

Boehm B. W., Brown J. R., Lipow M., « Quantitative Evaluation of Software Quality », *in* R. T. Yeh, C. V. Ramamoorthy (eds), *Proceedings of the* 2^{nd} *International Conference on Software Engineering*, ACM Press, p. 592-605, October, 1976.

Bouktif S., Amélioration de la prédiction de la qualité du logiciel par combinaison et adaptation de modèles, PhD thesis, Université de Montréal, Mai, 2005.

Briand L. C., Wüst J., « Empirical Studies of Quality Models in Object-Oriented Systems », *Advances in Computers*, vol. 59, p. 97-166, June, 2002.

Briand L., Devanbu P., Melo W., « An Investigation into Coupling Measures for C++ », *in* W. R. Adrion (ed.), *Proceedings of the* 19^{th} *International Conference on Software Engineering*, ACM Press, p. 412-421, May, 1997.

Brown W. J., Malveau R. C., Brown W. H., McCormick III H. W., Mowbray T. J., *Anti Patterns : Refactoring Software, Architectures, and Projects in Crisis*, 1^{st} edn, John Wiley and Sons, March, 1998.

Chidamber S. R., Kemerer C. F., « A Metrics Suite for Object Oriented Design », *IEEE Transactions on Software Engineering*, vol. 20, n° 6, p. 476-493, 1994.

Cohen W. W., « Fast effective rule induction », *Proceedings of the Twelfth International Conference on Machine Learning*, Morgan Kaufmann, p. 115-123, 1995.

Denier S., Guéhéneuc Y.-G., « Mendel : a Model, Metrics, and Rules to Understand Class Hierarchies », *in* R. K. Chris Verhoef, R. Lämmel (eds), 16^{th} *IEEE International Conference on Program Comprehension*, IEE Computer Society, june, 2008.

Dromey R. G., « Cornering the Chimera », *IEEE Software*, vol. 13, n° 1, p. 33-43, January, 1996.

Eden A. H., Kazman R., « Architecture, Design, Implementation », *in* L. Dillon, W. Tichy (eds), *Proceedings of the* 25^{th} *International Conference on Software Engineering*, ACM Press, p. 149-159, May, 2003.

Firesmith D., « A Hard Look at Quality Management Software », April, 2004.

Fowler M., *Refactoring – Improving the Design of Existing Code*, 1^{st} edn, Addison-Wesley, June, 1999.

Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns – Elements of Reusable Object-Oriented Software*, 1^{st} edn, Addison-Wesley, 1994.

Guéhéneuc Y.-G., « PTIDEJ : Promoting Patterns with Patterns », *in* M. E. Fayad (ed.), *Proceedings of the* 1^{st} *ECOOP workshop on Building a System using Patterns*, Springer-Verlag, July, 2005.

Guéhéneuc Y.-G., Antoniol G., « DeMIMA : A Multi-layered Framework for Design Pattern Identification », *Transactions on Software Engineering*, September, 2008. *Accepted for publication.*

Guéhéneuc Y.-G., Sahraoui H., Farouk Zaidi, « Fingerprinting Design Patterns », *in* E. Stroulia, A. de Lucia (eds), *Proceedings of the* 11^{th} *Working Conference on Reverse Engineering*, IEEE Computer Society Press, p. 172-181, November, 2004.

Harrison R., Counsell S. J., Nithi R. V., « Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems », *Journal of Systems and Software*, vol. 52, n° 2–3, p. 173-179, June, 2000.

Hitz M., Montazeri B., « Measuring Coupling and Cohesion in Object-Oriented Systems », *Proceedings of the* 3^{rd} *International Symposium on Applied Corporate Computing*, Texas A & M University, p. 25-27, October, 1995.

ISO, « International vocabulary of basic and general terms in metrology, international organization for standardization. », 1993.

Kerievsky J., *Refactoring to Patterns*, 1^{st} edn, Addison-Wesley, August, 2004.

Lange D. B., Nakamura Y., « Interactive Visualization of Design Patterns Can Help in Framework Understanding », *Proceedings of the* 10^{th} *annual conference on Object-oriented programming systems, languages, and applications*, ACM Press, p. 342 - 357, 1995.

Lorenz M., Kidd J., *Object-Oriented Software Metrics : A Practical Approach*, 1^{st} edn, Prentice-Hall, July, 1994.

Martin R. C., *Agile Software Development, Principles, Patterns, and Practices*, 2002.

McCall J. A., « Quality Factors », *Encyclopedia of Software Engineering*, vol. 1-2, p. 958-ff, December, 2001.

McNatt W. B., Bieman J. M., « Coupling of Design Patterns : Common Practices and Their Benefits », *in* T. Tse (ed.), *Proceedings of the* 25^{th} *Computer Software and Applications Conference*, IEEE Computer Society Press, p. 574-579, October, 2001.

Pressman R. S., *Software Engineering – A Practitioner's Approach*, 5^{th} edn, McGraw-Hill Higher Education, November, 2001.

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

Tegarden D. P., Sheetz S. D., Monarchi D. E., « A Software Complexity Model of Object-Oriented Systems », *Decision Support Systems*, vol. 13, n° 3–4, p. 241-262, March, 1995.

Venners B., « How to Use Design Patterns – A Conversation with Erich Gamma, Part I », May, 2005. `http://www.artima.com/lejava/articles/gammadp.html`.

Wendorff P., « Assessment of Design Patterns During Software Reengineering : Lessons Learned from a Large Commercial Project », *in* P. Sousa, J. Ebert (eds), *Proceedings of* 5^{th} *Conference on Software Maintenance and Reengineering*, IEEE Computer Society Press, p. 77-84, March, 2001.

Witten I. H., Frank E., *Data Mining : Practical Machine Learning Tools and Techniques with Java Implementations*, 1^{st} edn, Morgan Kaufmann, October, 1999.

Wydaeghe B., Verschaeve K., Michiels B., Damme B. V., Arckens E., Jonckers V., « Building an OMT-Editor Using Design Patterns : An Experience Report », *proceedings of the* 26^{th} *Technology of Object-Oriented Languages and Systems conference*, IEEE Computer Society Press, p. 20-32, August, 1998.

Patrons	Extensibilité(%)			Simplicité(%)			Généralité(%)			Modularité(%)			Mod. à l'exécution(%)		
	+	=	−	+	=	−	+	=	−	+	=	−	+	=	−
F. abstraite	**100,0**	0,0	0,0	**53,33**	13,33	33,33	**78,57**	21,43	0,0	**85,71**	7,14	7,14	46,15	38,46	15,38
Monteur	**90,91**	9,09	0,0	**54,55**	27,27	18,18	**54,55**	9,09	36,36	**90,91**	0,0	9,09	45,45	36,36	18,18
Fabrique	**75,0**	8,33	16,67	**58,33**	25,0	16,67	**58,33**	33,33	8,33	**66,67**	16,67	16,67	45,45	45,45	9,09
Prototype	**58,33**	33,33	8,33	**53,85**	30,77	15,38	**76,92**	15,38	7,69	38,46	**53,85**	7,69	**53,85**	38,46	7,69
Singleton	7,69	30,77	**61,54**	**92,86**	7,14	0,0	30,77	38,46	30,77	15,38	46,15	38,46	16,67	**50,0**	33,33
Adaptateur	**53,85**	38,46	7,69	46,15	38,46	15,38	**61,54**	30,77	7,69	**76,92**	15,38	7,69	**58,33**	33,33	8,33
Pont	**83,33**	16,67	0,0	18,18	27,27	**54,55**	**70,0**	0,0	30,0	**75,0**	16,67	8,33	**50,0**	33,33	16,67
O. composite	**100,0**	0,0	0,0	**69,23**	15,38	15,38	**76,92**	15,38	7,69	**71,43**	21,43	7,14	**53,85**	38,46	7,69
Décorateur	**91,67**	0,0	8,33	27,27	45,45	27,27	**90,91**	9,09	0,0	**72,73**	18,18	9,09	**72,73**	18,18	9,09
Façade	**53,85**	23,08	23,08	**83,33**	8,33	8,33	**50,0**	33,33	16,67	**50,0**	16,67	33,33	36,36	27,27	36,36
Poids-mouche	22,22	**44,44**	33,33	0,0	22,22	**77,78**	11,11	44,44	44,44	33,33	33,33	33,33	11,11	**66,67**	22,22
Proxy	41,67	**50,0**	8,33	**61,54**	23,08	15,38	**61,54**	30,77	7,69	**76,92**	15,38	7,69	38,46	**53,85**	7,69
Ch. de Resp.	**91,67**	8,33	0,0	**72,73**	9,09	18,18	**81,82**	18,18	0,0	**66,67**	16,67	16,67	**54,55**	27,27	18,18
Commande	**69,23**	15,38	15,38	46,15	15,38	38,46	**61,54**	23,08	15,38	**64,29**	21,43	14,29	41,67	41,67	16,67
Interpréteur	**66,67**	25,0	8,33	**50,0**	16,67	33,33	**64,29**	14,29	21,43	**58,33**	33,33	8,33	41,67	41,67	16,67
Itérateur	**92,31**	7,69	0,0	**69,23**	15,38	15,38	**72,73**	27,27	0,0	**57,14**	28,57	14,29	**50,0**	41,67	8,33
Médiateur	**58,33**	25,0	16,67	27,27	36,36	36,36	45,45	36,36	18,18	**50,0**	25,0	25,0	33,33	**50,0**	16,67
Memento	30,0	**50,0**	20,0	**50,0**	30,0	20,0	40,0	**50,0**	10,0	11,11	**66,67**	22,22	12,5	**75,0**	12,5
Observateur	**86,67**	6,67	6,67	**64,29**	21,43	14,29	**78,57**	7,14	14,29	**73,33**	20,0	6,67	**69,23**	23,08	7,69
État	**72,73**	18,18	9,09	**63,64**	9,09	27,27	**63,64**	18,18	18,18	**63,64**	9,09	27,27	**54,55**	18,18	27,27
Stratégie	**78,57**	14,29	7,14	**66,67**	26,67	6,67	**71,43**	21,43	7,14	**85,71**	7,14	7,14	**78,57**	14,29	7,14
P. de méthode	**80,0**	20,0	0,0	**66,67**	20,0	13,33	**73,33**	20,0	6,67	33,33	**53,33**	13,33	28,57	**64,29**	7,14
Visiteur	**66,67**	6,67	26,67	7,14	14,29	**78,57**	**78,57**	7,14	14,29	**73,33**	13,33	13,33	**57,14**	21,43	21,43

Tableau 5. *Évaluation par les répondants de l'impact des patrons de conception sur les attributs de qualité (I) (*$\mathtt{Positif} : +$, $\mathtt{Neutre} : =$, $\mathtt{Négatif} : -$*).*

Patrons	Fac. d'apprentissage(%)			Compréhensibilité(%)			Réutilisabilité(%)			Pass. à l'échelle(%)			Robustesse(%)		
	+	=	−	+	=	−	+	=	−	+	=	−	+	=	−
F. abstraite	35,71	28,57	35,71	38,46	30,77	30,77	**50,0**	42,86	7,14	21,43	**64,29**	14,29	0,0	**72,73**	27,27
Monteur	**54,55**	18,18	27,27	**81,82**	9,09	9,09	36,36	45,45	18,18	0,0	**81,82**	18,18	0,0	**90,0**	10,0
Fabrique	**50,0**	33,33	16,67	41,67	33,33	25,0	**63,64**	18,18	18,18	27,27	**54,55**	18,18	0,0	**88,89**	11,11
Prototype	46,15	23,08	30,77	**53,85**	23,08	23,08	**58,33**	8,33	33,33	33,33	**50,0**	16,67	9,09	**90,91**	0,0
Singleton	**85,71**	14,29	0,0	**92,86**	7,14	0,0	15,38	**53,85**	30,77	35,71	**57,14**	7,14	15,38	**69,23**	15,38
Adaptateur	**69,23**	7,69	23,08	46,15	23,08	30,77	**69,23**	23,08	7,69	0,0	45,45	**54,55**	0,0	**58,33**	41,67
Pont	27,27	36,36	36,36	**50,0**	33,33	16,67	41,67	16,67	41,67	10,0	30,0	**60,0**	10,0	**80,0**	10,0
O. composite	**76,92**	7,69	15,38	**69,23**	15,38	15,38	**61,54**	23,08	15,38	41,67	41,67	16,67	8,33	**91,67**	0,0
Décorateur	41,67	25,0	33,33	41,67	8,33	**50,0**	33,33	25,0	41,67	18,18	36,36	45,45	0,0	**77,78**	22,22
Façade	**66,67**	25,0	8,33	**75,0**	25,0	0,0	33,33	41,67	25,0	25,0	**50,0**	25,0	22,22	**77,78**	0,0
Poids-mouche	0,0	20,0	**80,0**	0,0	10,0	**90,0**	37,5	12,5	**50,0**	**77,78**	0,0	22,22	22,22	**66,67**	11,11
Proxy	46,15	38,46	15,38	30,77	46,15	23,08	**50,0**	33,33	16,67	27,27	36,36	36,36	10,0	**80,0**	10,0
Ch. de Resp.	**63,64**	36,36	0,0	33,33	33,33	33,33	**54,55**	27,27	18,18	10,0	**50,0**	40,0	25,0	**58,33**	16,67
Commande	38,46	30,77	30,77	30,77	30,77	38,46	33,33	16,67	**50,0**	25,0	33,33	41,67	33,33	**58,33**	8,33
Interpréteur	**66,67**	0,0	33,33	**66,67**	0,0	33,33	**54,55**	0,0	45,45	27,27	18,18	**54,55**	8,33	**83,33**	8,33
Itérateur	**50,0**	33,33	16,67	**50,0**	42,86	7,14	**69,23**	15,38	15,38	16,67	41,67	41,67	45,45	**54,55**	0,0
Médiateur	**54,55**	9,09	36,36	**58,33**	25,0	16,67	20,0	**50,0**	30,0	25,0	41,67	33,33	9,09	**81,82**	9,09
Memento	40,0	30,0	30,0	40,0	**50,0**	10,0	37,5	37,5	25,0	0,0	**50,0**	**50,0**	12,5	**62,5**	25,0
Observateur	**50,0**	14,29	35,71	40,0	33,33	26,67	**50,0**	28,57	21,43	16,67	**50,0**	33,33	9,09	**63,64**	27,27
État	**54,55**	18,18	27,27	**54,55**	0,0	45,45	20,0	40,0	40,0	18,18	45,45	36,36	10,0	**60,0**	30,0
Stratégie	**73,33**	20,0	6,67	**64,29**	28,57	7,14	46,15	30,77	23,08	21,43	42,86	35,71	33,33	**58,33**	8,33
P. de méthode	**53,33**	33,33	13,33	37,5	37,5	25,0	**57,14**	28,57	14,29	23,08	**61,54**	15,38	16,67	**75,0**	8,33
Visiteur	14,29	21,43	**64,29**	20,0	20,0	**60,0**	26,67	26,67	46,67	25,0	33,33	41,67	0,0	**72,73**	27,27

Tableau 6. *Évaluation par les répondants de l'impact des patrons de conception sur les attributs de qualité (II)* *(*$\texttt{Positif} : +$*,* $\texttt{Neutre} : =$*,* $\texttt{Négatif} : -$*).*

Patrons	Extensibilité(%)		Simplicité(%)		Généralité(%)		Modularité(%)		Mod. à l'exécution(%)	
	E	R(%)	E	R(%)	E	R(%)	E	R(%)	E	R(%)
F. abstraite	+	0,00	+	30,36	+	1,76	+	0,37	−	30,36
Monteur	+	0,15	+	30,36	+	30,36	+	0,15	−	30,36
Fabrique	+	1,76	+	30,36	+	30,36	+	5,92	−	30,36
Prototype	+	30,36	+	30,36	+	1,76	−	15,09	+	30,36
Singleton	−	0,15	+	0,15	−	5,92	−	0,37	−	0,37
Adaptateur	+	30,36	−	30,36	+	15,09	+	1,76	+	30,36
Pont	+	0,37	−	0,37	+	5,92	+	1,76	+	50,00
O. composite	+	0,00	+	5,92	+	1,76	+	5,92	+	30,36
Décorateur	+	0,15	−	5,92	+	0,15	+	5,92	+	5,92
Façade	+	30,36	+	0,37	+	50,00	+	50,00	−	15,09
Poids-mouche	−	1,76	−	0,00	−	0,15	−	5,92	−	0,15
Proxy	−	30,36	+	15,09	+	15,09	+	1,76	−	15,09
Ch. de Resp.	+	0,15	+	5,92	+	0,37	+	5,92	+	30,36
Commande	+	5,92	−	30,36	+	15,09	+	15,09	−	30,36
Interpréteur	+	5,92	+	50,00	+	15,09	+	30,36	−	30,36
Itérateur	+	0,15	+	5,92	+	5,92	+	30,36	+	50,00
Médiateur	+	30,36	−	5,92	−	30,36	+	50,00	−	5,92
Memento	−	5,92	+	50,00	−	30,36	−	0,15	−	0,15
Observateur	+	0,15	+	15,09	+	1,76	+	1,76	+	5,92
État	+	5,92	+	15,09	+	15,09	+	15,09	+	30,36
Stratégie	+	1,76	+	5,92	+	5,92	+	0,37	+	1,76
P. de méthode	+	0,37	+	5,92	+	1,76	−	5,92	−	5,92
Visiteur	+	5,92	−	0,15	+	1,76	+	1,76	+	30,36
	19 + / 4 −		16 + / 7 −		19 + / 4−		18 + / 5 −		11 + / 12 −	

Tableau 7. *Estimation de l'impact des patrons de conception sur les attributs de qualité (I).*

Patrons	Fac. d'apprentissage(%)		Compréhensibilité(%)		Réutilisabilité(%)		Pass. à l'échelle(%)		Robustesse(%)	
	E	R(%)	E	R(%)	E	R(%)	E	R(%)	E	R(%)
F. abstraite	−	15,09	−	15,09	+	50,00	−	1,76	−	0,00
Monteur	+	30,36	+	0,37	−	15,09	−	0,00	−	0,00
Fabrique	+	50,00	−	30,36	+	15,09	−	5,92	−	0,00
Prototype	−	30,36	+	30,36	+	30,36	−	5,92	−	0,15
Singleton	+	0,37	+	0,15	−	0,37	−	15,09	−	0,37
Adaptateur	+	5,92	−	30,36	+	5,92	−	0,00	−	0,00
Pont	−	5,92	+	50,00	−	30,36	−	0,15	−	0,15
O. composite	+	1,76	+	5,92	+	15,09	−	30,36	−	0,15
Décorateur	−	30,36	−	30,36	−	5,92	−	0,37	−	0,00
Façade	+	5,92	+	1,76	−	5,92	−	1,76	−	1,76
Poids-mouche	−	0,00	−	0,00	−	15,09	+	1,76	−	1,76
Proxy	−	30,36	−	5,92	+	50,00	−	5,92	−	0,15
Ch. de Resp.	+	15,09	−	5,92	+	30,36	−	0,15	−	1,76
Commande	−	15,09	−	5,92	−	5,92	−	1,76	−	5,92
Interpréteur	+	5,92	+	5,92	+	30,36	−	5,92	−	0,15
Itérateur	+	50,00	+	50,00	+	5,92	−	0,37	−	30,36
Médiateur	+	30,36	+	30,36	−	1,76	−	1,76	−	0,15
Memento	−	30,36	−	30,36	−	15,09	−	0,00	−	0,15
Observateur	+	50,00	−	30,36	+	50,00	−	0,37	−	0,15
État	+	30,36	+	30,36	−	1,76	−	0,37	−	0,15
Stratégie	+	1,76	+	15,09	−	30,36	−	1,76	−	5,92
P. de méthode	+	30,36	−	15,09	+	30,36	−	1,76	−	0,37
Visiteur	−	0,37	−	1,76	−	1,76	−	1,76	−	0,00
	14 + / 9 −		11 + / 12 −		11 + / 12−		1 + / 22 −		0 + / 23 −	

Tableau 8. *Estimation de l'impact des patrons de conception sur les attributs de qualité (II).*