Titre: Title:	Un modèle de qualité logicielle basé sur une architecture favorisant l'agrégation des données selon le processus d'analyse hiérarchique
Auteur: Author:	Marc-André Poulin
Date:	1998
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Poulin, M.-A. (1998). Un modèle de qualité logicielle basé sur une architecture favorisant l'agrégation des données selon le processus d'analyse hiérarchique [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6935/

UNIVERSITÉ DE MONTRÉAL

UN MODÈLE DE QUALITÉ LOGICIELLE BASÉ SUR UNE ARCHITECTURE FAVORISANT L'AGRÉGATION DES DONNÉES SELON LE PROCESSUS D'ANALYSE HIÉRARCHIQUE

MARC-ANDRÉ POULIN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
FÉVRIER 1998

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

UN MODÈLE DE QUALITÉ LOGICIELLE BASÉ SUR UNE ARCHITECTURE FAVORISANT L'AGRÉGATION DES DONNÉES SELON LE PROCESSUS D'ANALYSE HIÉRARCHIQUE

présenté par : POULIN Marc-André
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. GRANGER Louis, M.Sc., président
M. ROBILLARD Pierre N., Ph. D., directeur de recherche et membre
M. PLANCHE Rémi, M.Sc., membre

REMERCIEMENTS

Je remercie très sincèrement tous ceux et celles qui ont collaboré de près ou de loin à la réalisation de ce travail.

Toute ma gratitude va d'abord à monsieur Pierre N. Robillard qui m'a supporté dans toutes mes démarches au cours de ce projet.

Je tiens à remercier monsieur Benoît Guay, du groupe DMR, qui m'a offert l'opportunité de travailler sur un projet qui suscite beaucoup d'intérêt pour moi. Également, je remercie monsieur Jean Boisvert et madame Dominique Lesage, tous les deux de DMR, qui furent d'une aide inestimable tout au long du projet.

Je remercie également mes collègues de travail, messieurs Patrick D'astous, Philippe Mathieu, Jean-Sébastien Neveu, Martin Truchon et François Trudel, madame Hélène Béneteau de Laprairie ainsi que le personnel du laboratoire RGL, messieurs André Beaucage, Tommy Lapierre, Benoit Lefebvre et Carl Paquet qui font en sorte que l'environnement soit propice à l'épanouissement intellectuel.

Finalement, je tiens à remercier très chaleureusement mes proches. monsieur Jean-Yves Poulin qui a rendu ce document infiniment plus clair, monsieur Richard Poulin qui a suscité des interrogations qui m'ont permis d'éviter plusieurs embûches, monsieur Paul Poulin qui, par son humour, m'a fait comprendre qu'on pouvait s'amuser tout en travaillant, madame Rachel Larabie pour son support, sa motivation et sa joie de vivre et, finalement, madame Lise Rochette pour m'avoir enduré pendant tout ce temps.

Je m'en voudrais d'oublier de mentionner l'importante contribution du Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) au succès de ce projet de maîtrise.

Merci à tous.

RÉSUMÉ

Le but de ce projet de recherche est de développer un modèle de qualité logicielle qui servira lors de l'évaluation de la qualité des logiciels. L'élaboration du modèle suit une procédure rigoureuse qui se décompose en quatre étapes principales. La première étape est de choisir l'architecture sur laquelle le modèle est basé. L'architecture choisie appartient à la famille des *arborescences*. Ayant choisi l'architecture, les deux prochaines étapes consistent à la peupler, c'est-à-dire choisir les attributs externes et internes du modèle. Pour ce qui est des attributs externes, une douzaine de facteurs de qualité appartenant à deux domaines différents de la qualité ont été retenus. Ces facteurs de qualité sont liés à une quarantaine de critères de qualité qui ne sont pas chevauchés. Les attributs internes (métriques logicielles) ont été exclus du modèle puisqu'ils entraient en conflit avec les objectifs de notre modèle. Voulant développer un modèle à portée générale, il s'est avéré impossible de trouver des métriques s'appliquant à toutes les classes de logiciel. La dernière étape est le choix de la procédure d'agrégation des données, procédure qui permet d'agréger les données des niveaux inférieurs du modèle vers les niveaux supérieurs. La procédure d'agrégation retenue est le processus d'analyse hiérarchique. Les différentes vues, parfois contradictoires, de la qualité ont été expressément prises en considération dans la procédure d'élaboration. Ainsi, il existe donc une version du modèle pour les utilisateurs, les réalisateurs et le maître d'ouvrage.

ABSTRACT

The development goal of this research project is to design a new software quality model for use in assessing software quality. Software quality model development follows a rigorous process broken down into four main steps. The first step involves defining the architecture on which the model will be founded. The architecture chosen belongs to the arborescence family. The next two steps consists of populating the chosen architecture, which means defining the external and internal attributes of the model. Twelve quality factors belonging to two different fields of quality were retained as external attributes. The twelve quality factors are linked to some forty quality criteria that do not overlap. Internal attributes (software metrics) are not included in the model since they were in conflict with the model's objectives. For want of developing a model which is general in scope of application, it was not possible to find software metrics applicable to all software classes. The last step entails finding an appropriate data aggregation technique to roll up data from the bottom level of the model to its hightest level. The hierarchical analysis technique (AHP) was retained. Different views of quality, often contradictory, were purposely taken into consideration in the development process. The end result is that there is one version of the model for users, one for the developers and one for the managers.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

INTRODUCTION

La qualité est devenue un mot populaire de nos jours. Bien que sa popularité ait grandi exponentiellement au cours des dernières années, ce n'est pas un concept né d'hier. Selon Kitchenham et Walker (1986), il n'y a pas de gestionnaire, concepteur, vendeur ou inspecteur qui peut considérer son vocabulaire complet à moins qu'il ne contienne le mot qualité répété très souvent. Une étude récente, relatée par Garvin (1984), des départements d'affaires de compagnies majeures en Amérique du Nord, a révélé que les gestionnaires considèrent l'atteinte de haut niveau de qualité comme leur principale préoccupation.

Évidemment, le domaine du logiciel n'échappe pas à l'engouement créé par le phénomène de la qualité. Selon Kaposi et Kitchenham (1987), l'atteinte d'un haut niveau de qualité est un prérequis pour survivre dans un marché de grande compétitivité comme celui du logiciel. Une constatation encore plus frappante est qu'une des principales causes de la crise du logiciel, crise qui dure depuis plus de vingt ans, est considérée comme étant la faible qualité des produits logiciels. Les problèmes reliés à la qualité proviennent du fait que la plupart des systèmes logiciels échouent à rencontrer les attentes des clients. Ainsi, ils n'exécutent pas toutes les fonctions requises, ils ne sont pas fiables, ils sont difficiles à utiliser et ils n'atteignent pas les niveaux de performance exigés. Ces problèmes doivent être ajoutés aux plaintes usuelles, c'est-à-dire que les systèmes logiciels coûtent plus cher et sont plus longs à produire que ce à quoi les clients s'attendent. Même lorsque le logiciel est livré, il a souvent besoin de modifications substantielles avant qu'il puisse être parfaitement opérationnel. Les statistiques parlent d'elles-mêmes : le bureau général américain de comptabilité rapporte que pour 9 projets logiciels ayant coûté $6.8 millions, seulement 2% des logiciels ont été utilisés comme livrés (Dunn et Ullman, 1982).

La première interrogation qui nous vient à l'esprit lorsque nous avons reçu un produit livré à temps, dans les limites du budget et qui remplit efficacement et correctement toutes les fonctions qui lui ont été spécifiées est : *pourquoi considérer la qualité*

logicielle ? Selon Boehm et al. (1978), il y a plusieurs problèmes potentiels qui feront en sorte que nous ne serons pas satisfaits du produit reçu. Ces propos recoupent ceux de Cavano et McCall (1978) qui mentionnent que les systèmes logiciels possèdent plusieurs qualités ou attributs aussi importants pour l'utilisateur que les fonctions exécutées par ces systèmes.

Voici quelques-uns des problèmes que nous pouvons rencontrer :

- *le logiciel peut être difficile à comprendre et difficilement modifiable.* Ce qui mènera inévitablement à des coûts excessifs de maintenance logicielle. Sachant que la phase de maintenance compte pour environ 70% des efforts logiciels, il est inacceptable qu'un logiciel soit difficile à maintenir ;

- *le logiciel peut être difficile à utiliser ou peut mener à une utilisation impropre.* Un rapport de la GAO ("General Accounting Office") a identifié plus de $10,000,000 de frais gouvernementaux inutiles dus à des problèmes d'ADP ("Automatic Data Processing") dont la plupart étaient attribuables au fait que les logiciels menaient à une utilisation impropre ;

- *le produit logiciel peut être dépendant d'un certain type de machines ou difficile à intégrer avec d'autres systèmes.* Ce problème est déjà assez compliqué maintenant et avec la prolifération des types de machines, il deviendra de plus en plus complexe.

La majorité des gens de la communauté informatique s'entendent sur le fait que le manque de qualité logicielle est causé par un manque de contrôle sur les processus de développement et de maintenance logiciels. De toute évidence, l'amélioration de la qualité passe par l'amélioration de ces processus. Ainsi, de nombreuses techniques d'assurance qualité telles les revues, inspections, « walkthrough » ont émergé au cours des dernières années. Certes, ces techniques contribuent à améliorer la qualité logicielle, mais malheureusement leur emploi ne garantit pas hors de tout doute l'atteinte d'un haut niveau de qualité. Ainsi, il n'est pas suffisant de se fier uniquement

à la capacité d'un processus à réaliser un logiciel de qualité, il est essentiel d'évaluer réellement si la qualité du logiciel correspond à celle attendue.

Malheureusement, l'état de l'art dans l'évaluation de la qualité est encore très peu convaincant. Ce manque de maturité a comme principale source l'incapacité de modéliser adéquatement la qualité logicielle. Ainsi, selon Robillard (1993), les modèles de qualité existants ne sont pas toujours cohérents entre eux. Dans cet ordre d'idée, Dromey (1995) stipule qu'il n'y aura pas de gain de qualité tant qu'il n'y aura pas de modèles de qualité adéquats disponibles. Ce dernier a analysé différents modèles de qualité existants (Boehm et al., 1978 ; Bowen, 1976 ; Deutsh et Willis, 1988 ; Kitchenham, 1987 ; Kernighan et Plaugher, 1974 ; Nance, 1992) et il s'avère qu'aucun d'eux ne fait l'objet d'un consensus puisqu'à priori ils ne sont pas assez complets pour comprendre toutes les implications de la qualité logicielle. Selon ISO-9126 (1991), bien que les études sur la modélisation et l'évaluation de la qualité s'avèrent utiles, elles ont aussi causé la confusion à cause des nombreux aspects de la qualité offerts. C'est pour cette raison qu'un comité technique ISO/IEC a élaboré un modèle de qualité normalisé présenté dans le standard ISO-9126. Or, comme il sera démontré, ce modèle comporte également des lacunes.

Est-ce que le jeu en vaut la chandelle ? Est-il profitable d'essayer de comprendre l'essence de la qualité logicielle sachant que celle-ci est excessivement complexe et en est encore à ses premières armes ? La réponse semble être positive si on se fie au nombre croissant de publications sur la qualité logicielle. Selon Boehm et al. (1978), le premier bénéfice d'une capacité accrue de gérer les considérations relatives à la qualité logicielle serait une amélioration de l'efficacité des coûts de maintenance. Ainsi, l'insuffisance d'atteindre un certain niveau d'une qualité (i.e., maintenabilité) se transforme directement en trop de ressources consommées.

Le présent document s'intéresse, comme vous l'avez sûrement constaté, à l'évaluation de la qualité du logiciel. L'objectif ultime est de présenter le modèle de qualité logicielle qui a été développé, modèle qui est à la base de l'évaluation de la qualité du logiciel.

Le document est divisé en trois parties principales. La première partie, les chapitres 1 et 2, a pour objectif d'initier le lecteur aux deux composantes de l'évaluation de la qualité du logiciel, soit la qualité logicielle et l'activité de mesure (ou d'évaluation). La deuxième partie, les chapitres 3 à 7, s'intéresse à tous les aspects impliqués dans la modélisation de la qualité logicielle. La dernière partie, le chapitre 8, s'avère être le cœur du document puisque le modèle de qualité logicielle développé y est présenté.

Première partie : concepts généraux

CHAPITRE I
PHILOSOPHIE DE LA QUALITÉ

Traditionnellement, les informaticiens et les ingénieurs logiciels ont considéré que leurs objectifs étaient surtout de fournir un produit logiciel qui exécute certaines fonctions spécifiées d'avance. L'atteinte d'un certain niveau de qualité n'était pas, en général, une priorité bien définie.

Or, aujourd'hui, nous pouvons affirmer que la communauté informatique est préoccupée par la qualité logicielle et que les développeurs les plus expérimentés sont d'accord sur le fait que la qualité logicielle est un but important. Kitchenham (1987) va même jusqu'à affirmer que l'atteinte d'un haut niveau de qualité est devenue un prérequis pour survivre dans un marché de grande compétitivité comme celui du logiciel.

Nous aborderons dans ce chapitre la première composante de l'évaluation de la qualité du logiciel soit la qualité logicielle d'un point de vue général. En premier lieu, nous discuterons de la nébulosité entourant le concept de la qualité logicielle (section 1.1). Sachant que le concept en est un très complexe, vaut-il réellement la peine d'en tenir compte ? (section 1.2) Ayant présenté les difficultés et les bénéfices du concept, il est possible de s'attarder à une question qui demeure encore ouverte : en quoi consiste la qualité logicielle ? (section 1.4) Or, avant d'analyser les définitions existantes, nous verrons que ces définitions dépendent du point de vue qui est considéré (section 1.3).

1.1 Difficultés inhérentes au concept de la qualité logicielle

Selon Robillard (1993), un des principaux objectifs du génie logiciel est d'améliorer la qualité des produits logiciels. La tâche s'avère être excessivement ardue au dire de Cavano et McCall (1978), puisque nous ne sommes pas à ce jour capables de définir précisément en quoi consiste la qualité logicielle, comme nous le verrons à la section 1.4. Une des raisons de la difficulté du concept est que le logiciel a toujours été vu comme une abstraction. Contrairement au matériel, le logiciel n'a pas de présence physique.

Crosby (1979) discute d'une des principales difficultés liées au concept de la qualité logicielle :

> « Le problème de la gestion de la qualité n'est pas que ce que les gens ne connaissent pas. Le problème est ce qu'ils croient connaître...
>
> À cet égard, la qualité a beaucoup de choses communes avec le sexe. Chacun est en sa faveur (sous certaines conditions évidemment). Chacun croit qu'il comprend l'acte (même s'il ne voudrait pas avoir à l'expliquer). Chacun pense que son exécution est seulement une question de suivre les tendances naturelles (après tout, l'acte se fait d'une façon ou d'une autre). Et, évidemment, la plupart des gens croient que les problèmes dans ce domaine sont causés par les autres gens (si seulement ils pouvaient prendre le temps de faire les choses correctement).»

Cavano et McCall (1978), tant qu'à eux, illustrent, à l'aide d'exemples, les difficultés liées à l'évaluation de la qualité logicielle :

> "Si la *maintenabilité* d'un programme est pour être évaluée, quelqu'un pourrait élaborer l'hypothèse que plus le nombre de branches non conditionnelles d'un programme augmente, plus il sera difficile de maintenir le programme. Cependant, la forme exacte de la relation entre la *maintenabilité* et le nombre de

branches non conditionnelles n'est pas connue. Il peut s'agir d'une fonction monotone croissante et pour chaque branche non conditionnelle, le degré de difficulté pour la maintenance augmente par un certain delta ou la relation peut être de la forme d'une fonction par niveau où pour certaines valeurs seuils le degré de difficulté prend un saut d'un quantum. L'espoir est que les relations spécifiques soient découvertes et converties en des classements compréhensibles pour les qualités de haut niveau. Pour la *maintenabilité*, ce classement pourrait être en termes du nombre moyen de personnes-jours nécessaires pour corriger une erreur.

Considérons deux programmes, A et B, qui sont issus des mêmes spécifications, écrits dans le même langage et implantés sur le même ordinateur. Le programme A fonctionne 10% plus vite, a 5% moins de fautes sous les mêmes conditions de tests et coûte 20% moins cher que le programme B et est similaire pour ce qui est de la *maintenabilité* et des aspects de documentation. Quel programme est de meilleure qualité ? Une réponse impulsive serait de choisir le programme A. Cependant, comment quelqu'un peut-il être certain que les tests sur les deux programmes étaient vraiment identiques ? Et qu'est-ce-que 10% plus rapide signifie ? Peut-être que le programme B a été développé pour être exécuté dans la moitié de l'espace mémoire requis par le programme A. Maintenant, une interprétation complètement différente est possible ; avec une contrainte comme celle-ci, le programme B peut être considéré comme étant de meilleure qualité. Également, une autre interprétation peut arriver si une application différente était choisie."

Encore selon Cavano et McCall (1978), il est facile de s'apercevoir que l'évaluation de la qualité logicielle devient rapidement très difficile. Une des raisons est que la même fonction ou le même algorithme peuvent être implantés dans plusieurs formes et il n'est pas toujours clair de déterminer laquelle est la meilleure. Une autre raison est que la complexité et les interactions dans les développements de gros systèmes augmentent d'une façon non linéaire avec la taille.

Cavano et McCall (1978) font un parallèle intéressant entre l'évaluation de manuscrits et l'évaluation de la qualité logicielle : un logiciel peut être considéré comme étant composé presque uniquement de documentation. À partir de la définition des besoins jusqu'au code, le logiciel existe principalement sous la forme d'un document écrit. Or, nous savons qu'il y a peu de techniques prouvées pour déterminer la qualité des documents écrits.

1.2 Bénéfices d'une compréhension accrue du concept de la qualité logicielle

Boehm et al. (1978) fournissent un aperçu des situations dans lesquelles il est primordial d'avoir une bonne connaissance des diverses caractéristiques de la qualité logicielle :

- *Préparer les spécifications de qualité d'un produit logiciel*

  Formuler quelles sont les fonctions dont nous avons besoin et le niveau de performance désiré (vitesse, précision) est en soi assez simple. Indiquer que nous avons également besoin de maintenabilité et de compréhensibilité est également important, mais beaucoup plus difficile à formuler d'une façon qui peut être vérifiée.

- *Vérifier la conformité avec les spécifications de qualité*

  Cette vérification est essentielle pour que les exigences de qualité aient un certain sens. Elle peut être faite avec un grand investissement de ressources, mais ce genre de vérification est coûteux et dur sur le moral des gens.

- *Faire des compromis au niveau du design entre les coûts de développement et les coûts d'opération*

  Ces compromis sont spécialement importants puisque les budgets et/ou échéanciers serrés font souvent en sorte que la maintenabilité, la portabilité et l'utilisabilité sont escamotées.

- *Choix du système logiciel*

  Plusieurs utilisateurs ont besoin d'une évaluation de la facilité à laquelle chaque système peut être adapté aux besoins de leur organisation et à leur matériel existant.

Encore selon Boehm et al. (1978), le premier bénéfice d'une capacité accrue de gérer les considérations relatives à la qualité logicielle serait une amélioration de l'efficacité des coûts de maintenance. Ainsi, l'insuffisance d'atteindre un certain niveau d'une qualité (i.e., maintenabilité) se transforme directement en trop de ressources consommées.

1.3 Différentes vues de la qualité logicielle

La définition de la qualité dépend du point de vue qui est considéré. De ce fait, il existe plusieurs approches de définition de la qualité. La section 1.3.1 présente les cinq approches de Garvin (1984) et une description des implications organisationnelles de la concurrence de ces approches. Ces approches ne sont pas spécifiques au domaine du génie logiciel, mais elles s'y prêtent bien. La section 1.3.2 quant à elle présente les différentes vues de la qualité logicielle selon le standard ISO-9126 (1991) tout en faisant un parallèle avec celles de Garvin (1984).

1.3.1 Approche de définition de la qualité de Garvin (1984)

Garvin (1984) a identifié cinq approches majeures de définition de la qualité. Ces approches bien qu'elles proviennent de domaines (philosophie, économie, marketing et gestion des opérations) peu connexes avec le génie logiciel s'appliquent très bien à ce domaine.

Kaposi et Kitchenham (1987) fournissent un avant-goût de ces approches. Ainsi, selon eux, la qualité est une notion évasive. Dans son sens général, le terme peut référer à une propriété transcendantale d'un système qui est difficile à définir, impossible à mesurer mais facilement reconnaissable. Dans un contexte industriel, la qualité peut être vue comme une propriété émergente et simple qui mesure le degré selon lequel le système est adapté aux besoins et répond aux attentes du client. Le fournisseur considérera que la qualité n'est pas liée seulement au produit fini mais aussi à sa structure, son histoire, son processus de développement et ses attentes d'un point de vue technique, administratif et directionnel. La qualité signifie également de concevoir le système correctement et cela dès la première fois, faire un profit sur sa production et le livrer au client avec une grande confiance sur son succès à long terme.

Les grandes lignes de chacune des cinq approches apparaissent ci-dessous :

- la vue **transcendantale** fait correspondre la qualité avec "l'excellence innée". La qualité dans ce sens ne peut pas être définie de façon précise ; il s'agit de quelque chose qui est ressentie et non mesurée, reconnaissable seulement par l'entremise de l'expérience. C'est une approche qui ressemble quelque peu à l'appréciation de peinture et de musique. Lorsqu'une personne a manipulé, examiné, utilisée, analysé, considéré et a été exposée à plusieurs exemples de quelque chose, cette personne est en position pour reconnaître la présence ou l'absence de qualité, dans le sens transcendant, dans un autre exemple de cette chose ;

- la définition **basée sur le produit** est complètement différente et provient d'études en économie. Dans cette vue, la qualité est liée au contenu d'un produit, i.e. à la quantité d'ingrédients ou d'attributs possédés par le produit. Ceci implique qu'un niveau de qualité plus élevé peut seulement être atteint à un coût plus élevé et que la qualité est une caractéristique inhérente. Puisque la qualité dans cette vue reflète la présence ou l'absence d'attributs mesurables, elle peut être évaluée objectivement. Le classement de différents biens selon leur qualité est néanmoins incertain puisqu'il dépend du niveau d'accord sur les attributs que les clients préfèrent ;

- la vue **basée sur le client** de la qualité dépend également d'hypothèses sur ce que les clients désirent et peut être résumée comme "l'adaptation aux besoins". Ceci nous mène à deux questions. Premièrement, comment les nombreuses exigences individuelles des clients sont-elles composées en un énoncé définitif des besoins ? Deuxièmement, comment quelqu'un distingue-t-il entre les attributs du produit qui dénotent la qualité de ceux qui augmentent simplement la satisfaction du client? Deux produits peuvent être adaptés aux besoins, mais il y en a un qui fournit des caractéristiques additionnelles. Est-ce que le dernier est nécessairement de meilleure qualité ? L'intérêt principal dans cette vue de la qualité est clairement externe à l'organisation de production (*ou de développement logiciel*) ;

- l'approche **basée sur le processus** de la qualité peut être résumée par la "conformité aux spécifications". L'excellence correspond à rencontrer précisément les spécifications et cela dès la première tentative. Dans cette vue (qui n'est pas inconsistante avec la vue basée sur le client), deux produits très différents (une Rolls Royce et une Lada) peuvent atteindre le même degré de qualité. L'intérêt principal de la vue basée sur le processus est interne à l'organisation : elle se concentre sur la réduction des coûts en minimisant les déviations par rapport aux spécifications et, par le fait même, elle se concentre sur la réduction de la reprise du travail durant la production et par la suite sur les réparations lors de la maintenance ;

- l'approche **basée sur la valeur** est un composite des deux dernières approches. Elle définit la qualité comme l'habileté à fournir ce que le client veut à un prix acceptable et aussi à se conformer aux spécifications à un coût acceptable. Ainsi, la qualité (une mesure d'excellence) correspond à la valeur. Probablement la meilleure façon d'aborder cette approche est de trouver combien le client est prêt à payer et par la suite faire la conception pour que les coûts correspondent. Dans une autre forme, cependant, c'est l'approche utilisée par les gestionnaires lorsque la pression du coût et des échéanciers, le besoin de livrer un produit et l'état actuel du produit, commencent tous à entrer en conflit.

Le Tableau 1.1 présente des définitions représentatives de chacune des approches de définitions de la qualité. Il est à noter que la terminologie originale anglaise a été conservée pour ne pas introduire d'ambiguïtés.

Tableau 1.1 : définitions représentatives des approches de qualité de Garvin (1984)

Approches	Définitions
Transcendantale	• "Quality is neither mind nor matter, but a third entity indendent of the two ... even though Quality cannot be defined, you know what it is." (R. M. Pirsig, *Zen and the Art of Motorcycle Maintenance*, pp. 185, 213) • "... a condition of excellence implying fine quality as distinct from poor quality ... Quality is achieving or reaching for the highest standard as against being satisfied with the sloppy or fraudulent." (B. W. Tuchman, "The decline of Quality," *New York Times Magazine*, 2 November 1980, p.38)
Basée sur le produit	• "Differences in quality amount to differences in the quantity of some desired ingredient or attribute." (L. Abbott, *Quality and Competition*, pp. 126-127) • "Quality refers to the amounts of the unpriced attributes contained in each unit of the priced attribute." (K. B. Leffler, "Ambiguous Changes in Product Quality," *American Economic Review*, December 1982, p. 956)
Basée sur le client	• "Quality consists of the capacity to satisfy wants ..." (C. D. Edwards, "The meaning of Quality," *Quality Progress*, October 1968, p.37) • "Quality is the degree to which a specific product satisfies the wants of a specific consumer." (H. L. Gilmore, "Product conformance Cost," *Quality Progress*, June 1974, p. 16)

	• "Quality is any aspect of a product, including the services included in the contract of sales, which influences the demand curve." (R. Dorfman and P. O. Steiner, "Optimal Advertising and Optimal Quality," *American Economic Review*, December 1954, p. 831) • "In the final analysis of the marketplace, the quality of a product depends on how well it fits patterns of consumer preferences." (A. A. Kuehn and R. L. Day, "Strategy of Product Quality," *Harvard Business Review*, November-December 1962, p. 101) • "Quality consists of the extent to which a specimen (a product-brand-model-seller combination) possesses the service characteristics you desire." (E. S. Maynes, "The Concept and Measurement of Product Quality," in *Household Production Consumption*, p. 542) • "Quality is fitness for use." (J. M. Juran, ed., *Quality Control Handbook*, p 2-2)
Basée sur le processus	• "Quality (means) conformance to requirements." (P. B. Crosby, *Quality is Free*, p. 15) • "Quality is the degree to which a specific product conforms to a design or a specification." (Gilmore, June 1974, p. 16)
Basée sur la valeur	• "Quality is the degree of excellence at an acceptable price and the control of variability at an acceptable cost." (R. A. Broh, *Managing Quality for Higher Profits*, 1982, p.3) • "Quality means best for certain customer conditions. These conditions are (a) the actual use and (b) the selling price of the product." (A. V. Feigenbaum, *Total Quality Control*, p.1)

Selon Garvin (1984), la plupart des définitions existantes de la qualité appartiennent à une des catégories énumérées ci-dessus. La coexistence de ces approches a plusieurs implications importantes. Premièrement, cela permet d'expliquer les vues souvent concurrentes de la qualité à la fois des membres du département de marketing, de ceux de la production (processus) et de ceux de la recherche et du développement. Les gens du marketing prennent typiquement une approche basée sur le produit pour le sujet ; pour eux, un niveau de qualité plus élevé signifie de meilleures performances, des caractéristiques améliorées et d'autres améliorations qui augmentent le coût. Puisqu'ils voient le client comme le médiateur de la qualité, ils considèrent ce qui se passe lors de la production comme étant moins important que ce qui se passe sur le terrain. Les gens de la production *(équipe de développement logiciel)* prennent normalement une approche différente. Pour eux, la qualité signifie la conformité aux spécifications en mettant l'emphase sur « faire la bonne chose dès le premier essai ». Puisqu'ils associent un niveau de qualité bas avec un haut niveau de reprise du travail et de

matériel à jeter, ils s'attendent à ce qu'une amélioration de la qualité résulte en une réduction des coûts. Les gens de la recherche et du développement s'identifieront tant qu'à eux à la vue basée sur la valeur. Ils recherchent donc à élaborer ce que le client veut tout en se conformant aux spécifications et en respectant les contraintes de coût et de prix. Ces trois vues sont manifestement en conflit, et peuvent causer de sérieux torts au niveau de la communication. Des efforts de redressement peuvent devenir paralysés si la coexistence de ces deux perspectives n'est pas suffisamment reconnue. Malgré le potentiel de conflit, les compagnies doivent cultiver de telles perspectives différentes puisqu'elles sont essentielles à l'introduction réussie de produits de haute qualité. Se fier à une seule définition est une source fréquente de problème.

Comme résultat de ces différentes vues, la façon selon laquelle une compagnie est organisée peut avoir un impact majeur sur l'évaluation de la qualité et conséquemment sur les aspects de la qualité qui se verront accorder le plus d'importance (Kitchenham et Walker, 1986). Une compagnie qui est dominée par son groupe de marketing, par exemple, peut donner une emphase excessive sur la vue basée sur le client. Ceci détournera l'attention de la nécessité de s'assurer que la qualité en termes de production (processus) reçoive suffisamment d'attention. Cette omission peut mener à une mauvaise gestion du processus de production. Dans une autre compagnie, la recherche et le développement peuvent avoir une très grande importance. Ceci peut résulter en des produits trop élaborés qui voient le jour plus tard que prévu et qui ne répondent pas aux besoins du marché.

Encore selon Garvin (1984), les caractéristiques qui correspondent à la qualité doivent être premièrement identifiées à partir de recherches de marketing (une approche basée sur le client à la qualité) ; ces caractéristiques doivent par la suite être traduites en attributs de produits identifiables (une approche basée sur le produit à la qualité) ; et le processus de conception doit ensuite être organisé pour s'assurer que les produits sont conçus en accord avec les spécifications (une approche basée sur le processus à la qualité). Un processus qui ignore une de ces étapes échouera à concevoir des produits de qualité. Les trois vues sont nécessaires et doivent être minutieusement cultivées.

Comme nous le verrons au chapitre 3, ces étapes correspondent précisément à la structure générique des modèles de qualité.

Kitchenham et Walker (1986) transposent les dernières constatations de Garvin (1984) au domaine de la qualité logicielle. Ainsi, ils stipulent que la clé pour harmoniser les pressions conflictuelles des différents groupes de travail est le cycle de développement logiciel. Il y a un besoin de respecter et de répondre à chacune de ces différentes vues lors de la transition du produit dans son cycle de vie. De plus, les différents départements d'une organisation doivent savoir où et comment leurs vues de la qualité sont représentées. Ainsi, les exigences du produit doivent premièrement être établies (la vue du client) ; celles-ci doivent ensuite être converties en spécifications d'attributs (la vue basée sur le produit) ; et finalement le processus doit être mis en place pour produire le produit en accord avec les spécifications (la vue basée sur le processus). De cette façon, les exigences des parties intéressées peuvent toutes être adressées et codifiées et un cadre de gestion de la qualité peut être établi satisfaisant les différentes dimensions de la qualité.

Si ces vues de la qualité ont été balancées et harmonisées avec succès, il est fort probable que le produit final soit bien coté dans la vue transcendantale. Il est également probable que cette approche globale fournira la base aux gestionnaires pour former une vue basée sur la valeur lorsque la pression des compromis fait surface.

Néanmoins, selon Garvin (1984), chacune de ces approches majeures à la qualité partagent un problème commun. Chacune est vague et imprécise lorsque vient le temps de décrire les éléments à la base de la qualité des produits. L'analogie peut être portée aux modèles de qualité, qui ne se font pas trop bavards sur les méthodes précises de mesures de la qualité.

Garvin (1984) présente une étude intéressante des facteurs en corrélation avec la qualité logicielle. Ces facteurs sont le prix, la publicité, la part du marché, le coût et le rendement.

1.3.2 Différentes vues de la qualité logicielle selon le standard ISO-9126 (1991)

Dans le standard ISO-9126 (1991), trois vues distinctes de la qualité sont présentées. Ces vues sont :

- *Vue des utilisateurs* : les caractéristiques définies dans le standard ISO-9126 reflètent la vue des utilisateurs. Les utilisateurs sont principalement intéressés par l'utilisation du logiciel, par son fonctionnement et les résultats issus de celui-ci. Ils apprécient le logiciel sans en connaître les aspects internes et sans savoir comment il est réalisé. Les questions des utilisateurs peuvent se lire comme suit : Les fonctions exigées sont-elles disponibles dans le logiciel ? Dans quelle mesure le logiciel est-il fiable ? Dans quelle mesure le logiciel est-il efficace ? Le logiciel est-il facile à utiliser ? Dans quelle mesure le logiciel est-il facile à transférer sur un autre environnement ?

- *Vue des réalisateurs* : la réalisation impose que l'utilisateur et le réalisateur emploient les mêmes caractéristiques de qualité du logiciel, puisqu'elles s'appliquent depuis les exigences jusqu'à l'acceptation. Étant donné que les réalisateurs doivent produire un logiciel qui satisfera les exigences de qualité, ils sont concernés à la fois par la qualité du produit dans des états intermédiaires et par la qualité finale du produit ;

- *Vue du maître d'ouvrage* : habituellement, le maître d'ouvrage s'intéresse plus à la qualité globale qu'à une caractéristique particulière. La qualité est déterminée en pondérant chaque caractéristique, selon son besoin. Le maître d'ouvrage doit également résoudre des problèmes d'arbitrage. Par exemple, il peut devoir choisir entre l'amélioration de la qualité et la tenue des délais ou le dépassement de coût parce qu'il souhaite la plus grande qualité, dans les limites de coût de ressources humaines et de planification.

Les vues de la qualité du standard ISO-9126 (1991) semblent correspondre à celles de Garvin (1984). Ainsi, la vue des utilisateurs correspond à la vue basée sur le client, la vue des développeurs correspond à la vue basée sur le processus et la vue des gestionnaires correspond à la vue basée sur la valeur. Cependant, nous ne retrouvons pas de parallèles entre les deux premières vues de Garvin (la vue transcendantale et la vue basée sur le produit) et celles de ISO-9126.

1.4 En quoi consiste la qualité logicielle ?

Kitchenham et Walker (1986) stipulent qu'avant que l'on essaie de spécifier, mesurer, contrôler, assurer ou gérer la qualité, nous devons connaître quelle est sa signification. C'est ce que nous tenterons de réaliser dans cette section.

Un nombre élevé de définitions de la qualité logicielle apparaît dans la littérature. Ce qui constitue un véritable débat philosophique. Évidemment, certaines définitions, bien qu'elles ne soient pas l'objet d'un consensus, semblent être plus rigoureuses que d'autres. À ce titre, un plaisantin a déjà dit :

> "Chaque programme fait quelque chose de correcte, seulement ce n'est peut-être pas ce que nous voulons qu'il fasse".

Bien que cet énoncé semble ridicule, il soulève un point important : si une définition de la qualité logicielle ne tient pas compte de la satisfaction du client ou de la conformité aux exigences, il peut s'avérer qu'un logiciel, bien qu'il soit tout à fait inutile, soit considéré comme en étant un de qualité.

Une définition de la qualité logicielle qui est particulièrement intéressante a été proposée par Pressman (1987).

> "Conformité aux exigences explicites à la fois fonctionnelles et de performances, aux standards de développements explicitement documentés et aux caractéristiques implicites qui sont attendues de tous logiciels professionnellement développés."

Évidemment, cette définition peut être modifiée et étendue. Elle met l'emphase sur trois points importants :

1. Les exigences logicielles sont la fondation qui permet de mesurer la qualité. Un manque de conformité aux exigences résulte en un manque de qualité.

2. Les standards spécifiés définissent un ensemble de critères de développement qui guident la façon dont le logiciel est développé. Si ces critères ne sont pas suivis, il en résultera presque toujours un manque de qualité.

3. Il y a un ensemble d'exigences implicites qui souvent ne sont pas mentionnées. Si le logiciel répond aux exigences explicites, mais échoue à répondre aux exigences implicites, la qualité logicielle est suspecte.

Les organisations de standards telles « European Organisation for Quality Control (EOQC) », « American Society for Quality Control » et « International Organisation for Standardisation Technical Comittee 176 on Quality Assurance (ISO/ETC/176) » sont concernées à fournir des définitions formelles sur la qualité et ses sujets reliés. L'approche de définition de la qualité qui est généralement préconisée par ces organismes est de trouver un consensus entre l'opinion des fournisseurs et celle des clients sur la définition des concepts, standards et procédures liés à la qualité. Jusqu'à présent, selon Kaposi et Kitchenham (1987), ces efforts n'ont pas produit des résultats qui pourraient aider immédiatement l'industrie.

Ainsi, le standard ISO-9126 (1991) définit la qualité logicielle de la façon suivante :

> « Ensemble des traits et des caractéristiques d'un produit logiciel portant sur son aptitude à satisfaire des besoins exprimés ou implicites ».

Pour ce qui est du standard IEEE std. 1061-1992 (1992), la qualité logicielle correspond au degré selon lequel un logiciel possède une combinaison d'attributs désirés. Cette combinaison d'attributs doit être clairement définie ; sinon, l'évaluation de la qualité repose sur l'intuition. Pour ce standard, définir la qualité logicielle est équivalent à définir une liste d'attributs de qualité logicielle requis pour un système.

La qualité est souvent associée à des facteurs plus nébuleux tels le nom et la réputation du fournisseur. La perception de la qualité peut également être faussée par la publicité ou par des facteurs qui peuvent modifier les attentes des gens au lieu d'attributs visibles du système lui-même.

Dans ces circonstances, il n'est pas surprenant que la qualité soit dure à définir et que des standards de qualité soient difficiles à établir dans l'industrie. Ceci peut mener à une friction entre les clients et les fournisseurs parce que les attentes de qualité du client peuvent ne pas être captées dans les spécifications initiales en termes qui sont non ambigus, vérifiables objectivement à la livraison et exécutoires selon le contrat.

Selon Fenton (1991), la définition de la qualité choisie importe peu. Ce qui est important est de déterminer quels sont les attributs du produit logiciel qui présentent un intérêt pour l'utilisateur et aussi de savoir comment mesurer le degré d'accomplissement de ces attributs dans les produits logiciels. Lorsque nous y parviendrons, nous serons également capables de spécifier les attributs de la qualité d'une façon mesurable. Or, comme il est longuement discuté dans le chapitre 2, l'activité de mesure en est une ardue.

CHAPITRE II
PHILOSOPHIE DE L'ACTIVITÉ DE MESURE LOGICIELLE

La mesure est une activité excessivement importante comme en témoigne la célèbre maxime de Lord Kelvin vers la fin des années 1800 : "you cannot manage what you cannot measure". Malheureusement, le processus de mesure en est un très compliqué et souvent mal compris et ceci particulièrement dans le domaine logiciel.

Ce chapitre a donc pour objectif de présenter la deuxième composante de l'évaluation de la qualité du logiciel soit l'activité de mesure logicielle. Nous définirons tout d'abord en quoi consiste l'activité de mesure (section 2.1) et nous analyserons la situation actuelle de la mesure dans le domaine du logiciel (section 2.2). Ce qui nous amènera à discuter des objectifs du processus de mesure logicielle (section 2.3) ainsi que de l'utilité des mesures logicielles (section 2.4). Par la suite, nous catégoriserons les mesures de qualité logicielle (section 2.5). Finalement, nous présenterons la base fondamentale de toute mesure, soit la théorie de la mesure (section 2.6).

2.1 En quoi consiste l'activité de mesure logicielle ?

La définition formelle de l'activité de mesure est la suivante :

> « La mesure est un processus par lequel des nombres ou symboles sont assignés à des attributs d'entités dans le monde réel de telle sorte qu'ils sont décrits en accord avec des règles clairement définies. » (Fenton, 1991)

Il est à noter qu'une entité peut être un objet (ex : une personne, une chambre ou la phase de tests d'un projet) et qu'un attribut est une caractéristique de l'entité (ex : la pression d'une personne, la couleur d'une chambre ou la durée d'une phase de tests).

Plus spécifiquement, le domaine de mesure logicielle inclut les activités suivantes :

- modèles d'estimation et de mesures du coût et de l'effort ;
- mesures et modèles de productivité ;
- contrôle et assurance de la qualité ;
- collecte de données ;
- *modèles et mesures de qualité* ;
- modèles de fiabilité ;
- évaluation et modèles de performance ;
- complexité algorithmique/computationnelle ;
- métriques de complexité et de structures.

Évidemment, dans le présent document, nous sommes principalement concernés par les modèles et mesures de qualité. Pour ce qui est des autres activités, elles sont décrites en détail dans Fenton (1991).

2.2 Situation de l'activité de mesure logicielle

Les ingénieurs de la plupart des disciplines sont totalement conscients du rôle crucial du processus de mesure dans leurs activités. Malheureusement, au dire de Fenton (1991), une telle situation n'est pas réelle dans le domaine du génie logiciel. Fenton va même jusqu'à affirmer que les mesures logicielles sont un des sujets les plus mal compris dans le domaine plus général du génie logiciel. Ainsi, dans la plupart des cas :

- on échoue à définir des objectifs mesurables lors de la conception de produits logiciels. Par exemple, on promet que les produits seront fiables et

maintenables sans spécifier ce que cela signifie dans des termes mesurables. Gilb (1987) fournit l'assertion suivante : « projects without clear goals will not achieve their goals clearly » ;

- on échoue à mesurer les composantes qui affectent les vrais coûts des projets logiciels ;
- on n'essaie pas de quantifier la qualité des produits que l'on conçoit ;
- on se fie encore à des anecdotes pour nous convaincre d'utiliser une méthode de développement révolutionnaire ou un nouvel outil (ex : publicité à laquelle on se fie et qui n'a aucune base rigoureuse).

Ainsi, selon Fenton (1991), le peu de mesures qui sont effectuées sont exécutées de façon non fréquente, inconsistante et incomplète.

Tichy et al. (1995) vont encore plus loin en stipulant qu'une explication plausible de la déficience du domaine informatique (i.e., la crise du logiciel) est que les informaticiens (et aussi les ingénieurs logiciels), en général, ont négligé de développer des techniques de mesures adéquates.

Évidemment, tout au long du présent document, nous allons essayer de développer des mesures précises de la qualité logicielle. Malheureusement, en plus des dernières constatations, nous serons souvent frustrés par la nature subjective de l'activité. Cavano et McCall (1978) discutent de cette situation :

> « La détermination de la qualité est un facteur clé dans les événements de chaque jour - concours de dégustation de vins, événements sportifs (ex : gymnastique), concours de talent, etc. Dans ces situations, la qualité est jugée de la façon la plus fondamentale et directe qui soit : comparaison côte-à-côte d'objets sous les mêmes conditions avec des concepts prédéterminés. Le vin peut être jugé selon la clarté, la couleur, le goût, etc. Cependant, ce type de jugement est très subjectif ; pour avoir une certaine valeur, il doit être fait par un expert.

> La subjectivité et la spécialisation sont aussi appliquées pour déterminer la qualité logicielle. Pour aider à résoudre ce problème, une définition plus précise de la qualité logicielle est nécessaire et aussi une façon de dériver des mesures quantitatives de qualité logicielle pour une analyse objective. Une question majeure à ce point est de savoir si la qualité logicielle peut être mesurée. Plusieurs études prétendent (McCall et al. (1977), Boehm et al. (1978)) que la réponse à cette question est oui, mais il s'agit d'un oui qualifié. Puisqu'il n'y a rien de tel que la connaissance absolue, il ne faut pas s'attendre à mesurer la qualité logicielle d'une façon exacte étant donné que chaque mesure est partiellement imparfaite. Bronowsky (1973) décrit ce paradoxe de la connaissance de la façon suivante : « année après année, nous inventons des instruments plus précis avec lesquels nous observons la nature avec plus de raffinement. Et lorsque nous regardons nos observations, nous sommes décontenancés de voir qu'elles sont encore floues et nous sentons qu'elles sont plus incertaines que jamais. » »

En continuant sur les propos de Cavano et McCall (1978), nous pouvons faire l'analogie suivante :

> "Il y a deux façons de juger les performances d'un athlète vis-à-vis le sport qu'il pratique. Premièrement, si les performances sont mesurables d'une façon quantitative et directe (ex : course, saut en longueur, décathlon, etc.) il s'agit simplement d'utiliser des instruments de mesure (ex : chronomètre, ruban à mesurer, etc.) pour juger le niveau de performance atteint. À l'opposé, si les performances sont seulement mesurables d'une façon qualitative (ex : nage synchronisée, saut de plongeon, etc.), il faut avoir recours à des experts pour être en mesure de juger la performance. Bien attendu, il ne peut y avoir de controverse dans les sports « quantifiables ». Lorsqu'un coureur parcourt une distance plus rapidement qu'un autre, tout le monde s'entend sur le fait qu'il a offert une meilleure performance. En ce qui concerne les sports « non quantifiables de façon directe », il peut y avoir controverse. Ainsi, un juge peut trouver qu'une performance est meilleure qu'une autre alors que c'est exactement

> l'inverse pour un autre juge. Évidemment, une telle situation n'est pas souhaitable. Cependant, au fil des années, le jugement subjectif fait par un jury s'avère de plus en plus fiable. La raison est simple : on connaît mieux le sport qui est analysé. En revenant à ce qui nous concerne, nous remarquons que l'évaluation de la qualité logicielle se compare au deuxième type de sport soit celui « non quantifiable de façon directe ».''

2.3 Objectifs du processus de mesure logicielle

En affirmant que les mesures logicielles sont d'une certaine façon imprécise, Cavano et McCall (1978) font ressortir l'existence de zones d'incertitude dans la mesure. Ainsi, il semble nécessaire qu'un niveau de confiance soit établi pour permettre la tolérance dans la mesure logicielle. Toujours selon les mêmes auteurs, l'objectif réel de la mesure logicielle est de déterminer quelle est cette zone de tolérance et comment elle peut affecter l'utilisation de la mesure.

Au dire de Fenton (1991), les activités de mesure doivent avoir des objectifs ou des buts clairs et ce sont ces derniers qui détermineront le type d'entités et les attributs qui doivent être mesurés. Dans cette veine, Basili (1992) a défini le paradigme GQM (« Goal - Question - Metric »). En utilisant ce paradigme, les objectifs sont d'abord définis de façon à les rendre tangibles en les raffinant en un ensemble de questions quantifiables qui sont utilisées pour extraire l'information appropriée des modèles. Les questions et les modèles, à leur tour, sont utilisés pour définir un ensemble de métriques et de données à collecter et pour servir de cadre d'interprétation. Un modèle GQM peut être visualisé comme un arbre où la racine est occupée par un objectif, les noeuds par des questions et les feuilles par des métriques. Nous verrons ultérieurement que le paradigme GQM correspond très précisément à l'architecture d'un modèle de qualité logicielle.

Les objectifs diffèrent évidemment, dépendant du point de vue. Les objectifs des gestionnaires et des ingénieurs relativement à la mesure de la qualité logicielle sont les suivants :

- *gestionnaire* : a besoin de mesurer la qualité des produits logiciels qui sont développés dans un but de comparaison entre différents projets, faire des prédictions sur des projets futurs et établir des standards et des buts raisonnables pour l'amélioration ;

- *ingénieur* : a besoin de suivre la qualité des systèmes évoluants en faisant des mesures du processus et a besoin de spécifier les exigences de qualité en termes strictement mesurables.

2.4 Utilités des mesures logicielles

À partir des objectifs, on s'aperçoit qu'il y a deux grandes utilités de la mesure logicielle. Il y a, d'un côté, les mesures qui sont utilisées pour l'évaluation : elles aident à suivre le déroulement des projets logiciels tout en permettant un évaluation finale de la qualité. D'un autre côté, il y a les mesures qui sont utilisées principalement pour prédire le comportement futur des logiciels. Or, la portée de ces dernières mesures est encore douteuse. Ainsi, il apparaît que de telles prédictions soient plus ou moins fiables; c'est pourquoi nous nous intéresserons exclusivement aux mesures servant à l'évaluation.

2.5 Classification des mesures logicielles

Il y a trois classes d'entités en logiciel dont les attributs valent la peine d'être mesurés :

- *processus* : n'importe quelle activité reliée au logiciel ; normalement elles impliquent le facteur temps ;
- *produit* : n'importe quel artefact, livrable ou document qui provient de processus ;
- *ressources* : items qui servent d'entrées au processus.

Tout ce que l'on veut mesurer ou prédire en logiciel est un attribut d'une des trois entités énumérées précédemment.

Il y a une distinction à faire entre les attributs qui sont internes et ceux qui sont externes :

- les attributs *internes* d'une entité sont ceux qui peuvent être mesurés purement en terme de l'entité elle-même ;
- les attributs *externes* d'une entité sont ceux qui peuvent seulement être mesurés en respectant la façon dont l'entité est liée à son environnement.

Dans notre terminologie, les *attributs internes de qualité* correspondent aux *métriques logicielles de qualité* et les *attributs externes de qualité* correspondent aux *facteurs de qualité* et aux *critères de qualité.*

En utilisant cette structure de classement, nous présentons un aperçu des attributs présentant un intérêt pour chacune des entités (Tableau 2.1).

Tableau 2.1 : exemples d'attributs internes et externes

Produits	Grosseur Nombre de fautes Pourcentage de couverture des tests Nombre d'énoncés conditionnels	Fonctionnalité Fiabilité Utilisabilité Efficacité Maintenabilité Portabilité
Processus	Temps Effort Nombre de changements des exigences	Coût Stabilité
Personnel	Âge Prix	Productivité Expérience Fiabilité

Les attributs externes sont ceux que les gestionnaires et utilisateurs de logiciels veulent le plus prédire et mesurer. Malheureusement, ils sont les plus difficiles à mesurer. De plus, il n'y a généralement aucune définition de ces attributs qui fasse l'objet d'un consensus. Un attribut tel la qualité est tellement général qu'il est presque dénudé de sens (voir chapitre 1). Ainsi, nous sommes forcés de définir ces attributs en termes de certains autres attributs qui eux sont mesurables tels les attributs internes.

En plus du coût, les attributs externes des processus qui sont supposés être importants sont : la *contrôlabilité*, la *faculté d'observation* et la *stabilité*. Selon Fenton (1991), les gens croient que des attributs, tels ceux qui viennent d'être énumérés, ne sont pas assez maîtrisés pour permettre une mesure objective en accord avec les principes de la mesure qui sont décrits à la section 2.6. Ainsi, ces attributs sont normalement mesurés par des évaluations subjectives. L'espoir est que l'expérience acquise à faire de telles observations peut éventuellement être la base pour les relations empiriques requises pour la mesure objective.

Des exemples d'attributs externes des produits sont :

- *fiabilité* du code du programme. Ceci est dépendant à la fois de la machine sur laquelle le programme est exécuté et de son usage ;
- *compréhensibilité* d'un document de spécification. Ceci est dépendant des personnes qui essaient de comprendre le document ;
- *maintenabilité* du code source. Ceci est dépendant à la fois des personnes qui font la maintenance et des outils disponibles pour supporter cette activité.

D'autres exemples d'attributs externes des produits considérés importants sont : *utilisabilité*, *intégrité*, *efficacité*, *testabilité*, *réutilisabilité*, *portabilité* et *interfonctionnement*. Même si la tendance a été de considérer une quelconque représentation du code final comme étant le seul produit logiciel d'importance pour déterminer ces attributs externes, il est maintenant clair qu'il y a d'autres produits logiciels qui sont tout aussi importants.

Un attribut qui présente un grand intérêt et qui est applicable à toutes les ressources est le *coût*, qui est considéré dans beaucoup de situations comme étant dépendant de plusieurs autres attributs en plus de celui qui est le plus facilement mesurable, soit le prix du marché. Des entités qui sont considérées comme des ressources pour certains processus sont clairement des produits pour d'autres processus (exemples : les outils logiciels). C'est dans de telles situations que le coût d'une ressource doit prendre en considération non seulement le prix mais aussi des mesures d'attributs qui sont importants pour l'entité vue comme un produit (exemple : *fiabilité*, *utilisabilité*).

Or, dans le présent document, nous sommes principalement intéressés par les entités de la classe produit puisque notre but principal est d'évaluer la qualité des produits logiciels.

2.6 Théorie de la mesure

Cette description de la théorie de la mesure a été adaptée à partir de Fenton (1991).

La théorie de la mesure est maintenant reconnue comme étant la base scientifique nécessaire pour développer et raisonner sur n'importe quel type de mesure. Ainsi, à l'aide de cette théorie, il est possible de déterminer quelles sont les recherches valides et aussi de réaliser des recherches valides.

Cette théorie prétend répondre aux questions suivantes :

- jusqu'à quel point doit-on connaître un attribut avant qu'il soit possible de le mesurer ?
- comment savons-nous si nous avons vraiment mesuré l'attribut que nous désirons mesurer ?
- en utilisant la mesure, quels types d'énoncés valides (ayant un sens) peuvent être faits sur les attributs et les entités qui possèdent ces attributs ?
- quels types d'opérations pouvons-nous effectuer sur les mesures qui aient un sens ?

Le cadre permettant de répondre aux deux premières questions est fourni par la *condition de représentation* pour la mesure (section 2.6.2). Les réponses aux deux dernières questions sont liées à la compréhension de la notion d'échelle de mesure (section 2.6.3, 2.6.4 et 2.6.5). Toute activité de mesure logicielle sérieuse doit connaître les réponses aux quatre questions précédentes. Avant d'entrer dans les détails de la théorie de la mesure, nous allons commencer par l'introduire d'une façon générale (section 2.6.1).

2.6.1 Introduction à la théorie de la mesure

Pour mesurer quelque chose, il faut connaître les entités mesurées et il faut avoir une idée de l'attribut de l'entité que l'on essaie de mesurer numériquement. Même si ce point semble être évident, il est souvent ignoré dans beaucoup d'activités de collecte de données. Lorsque l'attribut et la façon de le mesurer ont été identifiés, l'accumulation de données est maintenant possible. L'analyse du résultat de ce processus conduit normalement à la clarification et à la réévaluation de l'attribut, ce qui améliore la précision de la mesure.

On distingue deux types de mesure :

- la mesure *directe* d'un attribut est une mesure qui ne dépend de la mesure d'aucun autre attribut ;
- la mesure *indirecte* d'un attribut est une mesure qui implique la mesure d'un ou plusieurs autres attributs.

Il y a une relation entre les attributs internes et les attributs externes d'un côté et la mesure directe et la mesure indirecte de l'autre côté. Selon les propos de Fenton (1991), les attributs internes peuvent en principe être mesurés directement. Cependant, ceci n'exclut pas la possibilité que des mesures plus précises puissent être obtenues indirectement. Les attributs externes ne peuvent pas être mesurés directement ; ils sont donc définis en termes d'attributs internes qui eux sont mesurables.

2.6.2 Théorie représentationnelle de la mesure

La théorie représentationnelle de la mesure présente un cadre mathématique qui est basé sur des ensembles, relations, axiomes et fonctions. Les composantes clés sont :

- *système de relation* : détermine les propriétés (ou axiomes) sur l'attribut qui capte une compréhension intuitive ou des observations empiriques. Ces propriétés sont souvent caractérisées par des relations sur l'ensemble des entités ;

- *théorème de la représentation* : démontre que l'attribut peut être représenté dans un système numérique approprié par une correspondance des entités qui préserve les propriétés et les relations ;

- *théorème de l'unicité* : démontre que n'importe quelle fonction d'entité à des nombres qui représentent l'attribut sont liés d'une certaine façon.

En général, l'attribut est caractérisé par l'ensemble des relations empiriques *R*. Cet ensemble, avec l'ensemble des entités *C*, est appelé *système de relations empiriques*, c'est-à-dire ***C***=(*C*,*R*). L'activité de mesure d'un attribut doit absolument être effectuée en respectant un système de relations empiriques particulier. Par exemple, l'ensemble des entités pourrait être un ensemble de personnes (l'ensemble *C*), l'attribut présentant un intérêt serait la grandeur et un exemple de relations appartenant à l'ensemble des relations empiriques (l'ensemble *R*) pourrait être R_1 : $(x,y) \varepsilon R_1$ si « x est plus grand que y ».

Ayant trouvé un système de relations empiriques pour un attribut, nous devons trouver un système numérique, comme l'ensemble des nombres réels $\Re$. Plus généralement, nous avons besoin d'un *système de relations numériques* qui consiste en un ensemble avec une ou plusieurs relations sur cet ensemble. L'idée est que pour chaque relation dans le système de relations empiriques d'un certain attribut, nous avons besoin d'une relation correspondante dans le système de relations numériques. En reprenant l'exemple précédent, une relation correspondante à la relation R_1 serait P_1 : $(x,y) \varepsilon P_1$ si $x > y$ (pour correspondre à « x est plus grand que y »).

En général, un système de relations numériques ***N*** consiste en un ensemble *N* avec un ensemble des relations *P* sur *N*, c'est-à-dire ***N***=(*N*,*P*).

Normalement, *M* est une *mesure pour un attribut* si elle est une correspondance d'un système de relations empiriques (*C*,*R*) dans un système de relations numériques (*N*,*P*). La correspondance *M* fait correspondre des entités dans *C* en des nombres i.e., des entités dans *N* et fait également correspondre chaque relation dans *R* en une relation dans *P*. De plus, il faut insister sur le fait que toutes les relations empiriques sont préservées dans le système de relations numériques. Cette dernière condition est appelée *condition de représentation*. Une correspondance *M* satisfaisant la condition de représentation est appelée un *homomorphisme* ou une *représentation*.

2.6.3 Échelles

Comme discuté précédemment, une mesure d'un attribut a été accomplie si nous pouvons assigner une représentation *M* à partir d'un système de relations empiriques observé *C* à un certain système de relations numériques *N*. Si nous avons une telle représentation, alors le triplet (*C*,*N*,*M*) est appelé une *échelle* où *M* réfère à l'échelle.

Une des questions importantes concernant les représentations et les échelles est la suivante : que faisons-nous lorsque nous avons différentes représentations possibles (et donc plusieurs échelles) dans le même système de relations numériques ? Cette question soulève le *problème de l'unicité*. En général, pour un système de relations donné, n'importe quelle correspondance qui transforme une représentation *M* dans une autre représentation *M'* est appelée une *transformation admissible* ; par conséquent les représentations *M* et *M'* sont aussi valides l'une que l'autre.

La classe des transformations admissibles pour un certain système de relations empiriques définit l'unicité de chacune des échelles et peut être utilisée pour définir le type d'échelle. Ce qui veut dire que le type d'échelle est dépendant des sortes de mises à l'échelle qui sont permises. Moins il y a de mises à l'échelle possibles pour un

certain système de relations, plus le type d'échelle sera restrictif (donc plus précis). Les types d'échelle les plus importants sont donnés dans le Tableau 2.2.

Tableau 2.2 : types d'échelle pour la mesure

Transformations admissibles	Type d'échelle	Exemples
$M'=F(M)$ (F : correspondance 1-1)	Nominale	Étiquetage/classification d'entités
$M'=F(M)$ (F : monotonique croissante i.e. $M(x)\geq M(y)$ $\Rightarrow M'(x)\geq M'(y)$)	Ordinale	Préférence, dureté, qualité de l'air
$M'=\alpha M+\beta\ (\alpha>0)$	Intervalle	Temps (calendrier), température (fahrenheit)
$M'=\alpha M\ (\alpha>0)$	Ratio	Intervalle de temps, longueur
$M'=M$	Absolue	Comptage

En terme général, n'importe quel attribut qui impose seulement une classification sur un ensemble d'entités a un type d'échelle *nominale* et n'importe quel attribut qui impose seulement un ordonnancement linéaire a un type d'échelle *ordinale*. Si, en plus de l'ordonnancement linéaire, il y a la notion de distance relative entre les entités, alors le type d'échelle est *intervalle*. S'il y a également la notion du zéro, alors le type d'échelle est *ratio*. Les échelles *absolues* surviennent lorsque les attributs sont seulement un décompte des entités.

2.6.4 La notion de validité

Maintenant que nous avons établi une théorie sur les types d'échelle, on peut l'utiliser pour décrire la notion de validité, ce qui nous fournit un mécanisme formel par lequel on peut analyser la véracité (le sens) des énoncés sur les mesures. Ceci nous fournira également une indication des limitations en rapport aux sortes de manipulations mathématiques pouvant être effectuées sur les nombres. Même si nous pouvons toujours effectuer des opérations comme l'addition, la soustraction et la moyenne, la

question clé est la suivante : ayant effectué ces opérations, peut-on déduire des énoncés valides sur les entités ayant été mesurées ?

N'importe quel énoncé sur la mesure devrait spécifier quelle échelle est utilisée. Le mécanisme formel par lequel on peut analyser la véracité est donné par la directive suivante :

> « Un énoncé impliquant une échelle numérique a un sens si sa véracité ou sa fausseté demeure inchangée si chaque échelle *M* impliquée est remplacée par une mise à l'échelle *M'* acceptable ».

Une des applications les plus puissantes de la théorie sur les types d'échelle et de la notion de validité est de déterminer quels types d'opérations ou analyses statistiques peuvent être appliqués sur des types particuliers de mesures. Le Tableau 2.3 présente ces informations de façon abrégée.

Tableau 2.3 : statistiques associées aux échelles de mesure

Échelle	Relations de définition	Exemples de transformations autorisées	Tests statistiques appropriés
Nominale	1. Équivalence	• Mode • Fréquence • Coefficient de contingence	Tests statistiques non paramétriques
Ordinale	1. Équivalence 2. Plus grand que	• Médiane • Percentile • Kendall τ • Spearman r_S • Kendall W	
Intervalle	1. Équivalence 2. Plus grand que 3. Ratio connu d'un intervalle	• Moyenne • Déviation standard • Corrélation produit-moment de Pearson • Multiple corrélation produit-moment	Tests statistiques non paramétriques et paramétriques

[illegible]	[illegible]	[illegible]	[illegible]
Ratio	1. Équivalence 2. Plus grand que 3. Ratio connu d'un intervalle 4. Ratio connu de deux valeurs d'échelle	• Moyenne géométrique • Coefficient de variation	

2.6.5 Mesures indirectes et notion de validité

Bien que la majorité des attributs puissent être mesurés directement de la façon qui a été présentée dans les dernières sections, il arrive que des mesures plus précises peuvent être prises *indirectement*. Ceci est réalisé en définissant de nouvelles échelles en fonction des anciennes. Également, pour ce qui concerne les modèles de qualité, la mesure indirecte est essentielle puisqu'il n'existe aucune mesure directe qui peut prétendre capter l'information des attributs de haut niveau de la qualité (facteur de qualité).

En général, on peut dire qu'une fonction M est une échelle indirecte si elle est définie en terme de certaines fonctions $M_1,M_2,...,M_n$ (pour n>0). Normalement, M et les M_i vont satisfaire une certaine condition $E(M_1,M_2,...,M_n,M)$. Généralement, E sera une équation liant M à $M_1,M_2,...,M_n$.

Comme pour les mesures directes, il faut trouver quelles sont les transformations admissibles de M qui préservent la condition E. La définition ci-dessous formalise ces dires :

> "Supposons que $E(M_1,M_2,...,M_n,M)$ tient pour l'échelle indirecte M. On dit que M' est une mise à l'échelle de M s'il y a des mises à l'échelle $M'_1,M'_2,...,M'_n$ de $M_1,M_2,...,M_n$ respectivement, tel que $E(M'_1,M'_2,...,M'_n,M')$ tient."

Deuxième partie : concepts spécifiques à la modélisation de la qualité logicielle

CHAPITRE III
INTRODUCTION À LA MODÉLISATION DE LA QUALITÉ LOGICIELLE

Ce chapitre vise à fournir une impression sur les concepts généraux relatifs aux modèles de qualité logicielle. Comme nous en avons déjà discuté dans la première partie du document, toute évaluation est basée sur un modèle d'où l'importance des modèles de qualité logicielle dans le contexte de l'évaluation de la qualité du logiciel.

Nous présentons en premier lieu ce en quoi consiste un modèle de qualité logicielle (section 3.1) ainsi que la description des deux types de modèles existants (général et spécifique) (section 3.3). Par la suite, nous discutons de l'état de l'art dans ce domaine (section 3.2). Après, nous utilisons l'architecture facteur-critère-métrique pour exposer les principaux concepts à la base de la modélisation de la qualité logicielle (section 3.4). Ayant introduit le sujet, nous verrons diverses procédures d'élaboration des modèles de qualité logicielle ainsi que les aspects particuliers à considérer lors de cette élaboration (section 3.5). Finalement, les principales utilités de tels modèles sont exposées (section 3.7). L'annexe II quant à elle présente une dizaine de modèles de qualité logicielle courants.

3.1 Qu'est-ce qu'un modèle de qualité logicielle ?

Pfaltz (1977) mentionne que beaucoup des phénomènes de la vie commune ont une existence bien connue mais qu'il est difficile, voir impossible, de les définir précisément. Comme nous l'avons vu au chapitre 1, il est clair que la qualité logicielle est un exemple typique de ces phénomènes. Or, un outil des plus puissants de l'approche scientifique est la substitution d'un modèle à la place du phénomène lui-même.

D'une façon générale, les modèles sont conçus pour faire ressortir certaines propriétés d'un système tout en négligeant pour des raisons de clarté certaines autres propriétés. Plus formellement, deux définitions distinctes d'un modèle sont :

> Un *modèle* est une représentation abstraite d'un objet. (Fenton, 1991)
> Un *modèle* est une représentation simplifiée d'un certain système. (Kaposi et Kitchenham, 1987)

Ainsi, les modèles s'avèrent d'une aide inestimable dans les mains du designer ou du gestionnaire qui, sans ces derniers, peuvent devenir confus par la masse de détails du système dans son intégralité.

Les définitions présentées précédemment sont très générales et suggèrent qu'il y a une multitude de classes différentes de modèles. Or, au dire de Fenton (1991), pour la mesure logicielle, il y a essentiellement deux types de modèles présentant un intérêt particulier :

- Les modèles qui sont des représentations abstraites des produits, processus et ressources. Nous en avons besoin pour définir les mesures d'une façon non ambiguë. De tels modèles doivent capter les attributs mesurés ; par exemple un modèle de flot de contrôle du code source peut capter certains attributs liés à la structure de contrôle du programme. Même pour définir

des mesures d'apparence simple comme la longueur du code source, nous avons besoin d'un modèle formel du code source d'un programme ;

- Les modèles qui sont des représentations abstraites des relations entre les attributs d'entités. Ces modèles lient normalement deux mesures ou plus dans une formule mathématique. C'est précisément à ce type de modèle que les modèles de qualité correspondent.

En se servant des dernières constatations, nous pouvons élaborer une définition d'un modèle de qualité logicielle :

> **"Un modèle de qualité logicielle est une représentation abstraite et simplifiée de tout ce qui touche ou affecte la qualité logicielle."**

À partir de maintenant, nos discussions seront concentrées sur les modèles de qualité et non pas sur le phénomène de la qualité lui-même et c'est à partir des modèles que nous tirerons des conclusions. Or, comme nous le verrons à la section 3.2, nous sommes obligés d'admettre que les modèles de qualité sont des approximations plus ou moins fidèles du phénomène de la qualité logicielle.

3.2 État de l'art dans la modélisation de la qualité logicielle

Avant d'entrer dans le vif du sujet, il est opportun de présenter les grandes lignes de l'évolution d'une discipline. Selon Robillard (1993), au début, il s'agit d'études individuelles. Par la suite, certaines de ces études se rejoignent sur des aspects communs et finalement des normes sont établies. La Figure 3.1 schématise ces dernières considérations. Or, malheureusement, la modélisation de la qualité est encore au stade des modèles individuels. Même s'il existe un modèle normalisé (ISO-9126), nous ne pouvons pas considérer que la discipline est au stade d'un modèle normalisé

puisque ce modèle considère seulement les attributs de haut niveau de la qualité et qu'il néglige volontairement les métriques logicielles.

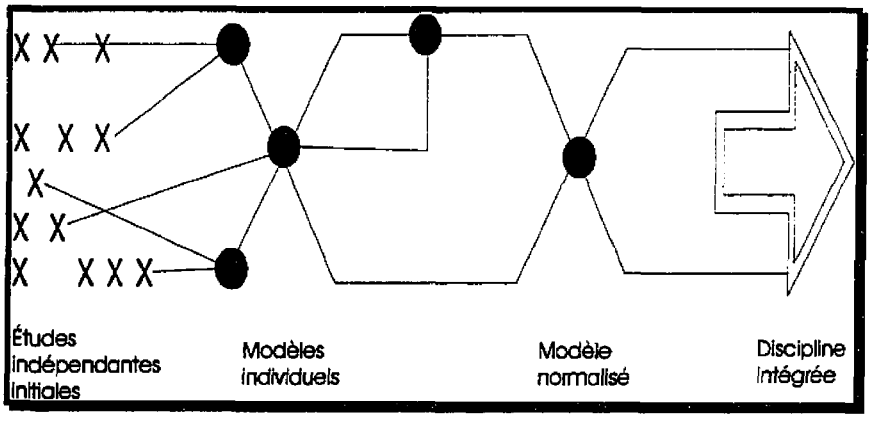


Figure 3.1 : évolution d'une discipline

Dromey (1995) quant à lui a analysé plusieurs modèles de qualité logicielle (Boehm et al., 1978 ; Bowen, 1976 ; Deutsh et Willis, 1988 ; Kitchenham, 1987 ; Kernighan et Plaugher, 1974 ; Nance, 1992) et il s'avère qu'aucun d'eux ne fait l'objet d'un consensus puisqu'à priori ils ne sont pas assez complets pour comprendre toutes les implications de la qualité logicielle. De plus, Dromey (1995) stipule que la plupart des modèles de qualité ne manipulent pas adéquatement les caractéristiques des produits et échouent à faire un lien direct entre les facteurs de qualité et les caractéristiques correspondantes du produit. Cavano et McCall (1978) sont en accord avec la dernière allégation de Dromey puisqu'ils stipulent que les relations entre les attributs qui composent la qualité sont très peu comprises.

Robillard (1993) affirme que les modèles de qualité logicielle existants ne sont pas toujours cohérents entre eux. Le même terme peut avoir des définitions distinctes ou encore des termes différents peuvent avoir la même définition. Devant la difficulté d'évaluer et de comparer la qualité d'un logiciel, la normalisation est devenue une nécessité selon Robillard (1993). Or, les organismes de normalisation, tels ISO, renvoient la balle dans le camp des chercheurs en stipulant que la maturité des modèles,

des termes et des définitions ne permet pas de les inclure dans un standard. Il est à noter que l'allégation de ISO-9126 (1991) vise les sous-caractéristiques et métriques des modèles de qualité logicielle.

Pressman (1992) offre une brève explication de la cause des déficiences des modèles de qualité ; ainsi, il stipule que l'élément qui complique les choses est la relation précise entre les variables qui sont mesurées et la qualité du logiciel.

3.3 Types de modèles de qualité logicielle

Il existe deux types distincts de modèles de qualité logicielle, soit les modèles généraux et les modèles spécifiques.

Les modèles généraux sont développés pour être utilisés avec toutes les classes d'applications logicielles existantes (logiciels de système, logiciels en temps réel, logiciels de gestion, etc.). Ainsi, les attributs de ces modèles sont choisis pour être applicables à n'importe quels logiciels.

À l'opposé, les modèles spécifiques sont développés pour être utilisés exclusivement avec une classe d'application logicielle en particulier. Les attributs de ces modèles sont donc choisis pour couvrir tous les aspects de la qualité qui sont pertinents à la classe considérée.

La majorité des auteurs tel ISO-9126 (1991), s'entendent sur le fait qu'il est excessivement ardu d'élaborer un modèle de qualité applicable à toutes les classes de logiciel puisque l'importance de chacune des caractéristiques des modèles dépend de la classe du logiciel considérée. Par exemple, la *fiabilité* est la caractéristique la plus importante pour les systèmes critiques; pour les systèmes en temps réel, c'est l'*efficacité* qui est déterminante et, pour les systèmes interactifs, l'*utilisabilité* domine.

Sachant que l'importance des attributs de qualité dépend de la classe d'application logicielle considérée, l'idéal serait d'élaborer un modèle spécifique pour chacune des classes de logiciels. Malheureusement, une telle activité demande beaucoup de ressources et de temps puisqu'il faut élaborer près d'une dizaine de modèles de qualité différents.

L'étude des modèles de qualité courants nous a permis de constater qu'aucun d'entre eux ne fait partie de la catégorie des modèles spécifiques. Ceci peut être en partie expliqué par le fait que la majorité des entreprises et des universités sont concernées par plusieurs classes d'applications et qu'il s'avère trop coûteux de développer un modèle pour chacune de ces classes.

3.4 Modèle de qualité logicielle basé sur l'architecture facteur-critère-métrique

Selon Fenton (1991) et Kitchenham (1987), les modèles de qualité logicielle sont généralement basés sur l'idée qu'il y a un certain nombre de *facteurs de qualité* des produits logiciels que l'on aimerait mesurer. Ces facteurs sont généralement consistants avec le point de vue de l'utilisateur de produits logiciels (voir section 1.3). Cavano et McCall (1978) mentionnent que ces facteurs sont également appropriés pour les gestionnaires pour être utilisés comme guides lors de la spécification des objectifs de qualité pour leurs systèmes logiciels. Il y a une hypothèse implicite stipulant que les facteurs de qualité (comme l'attribut *qualité* lui-même) sont d'un niveau trop élevé pour être significatifs ou mesurés directement. Ces derniers doivent donc être décomposés en attributs de plus bas niveau normalement appelés *critères de qualité* qui correspondent à la vision de la qualité des développeurs de systèmes. Un autre niveau de décomposition est requis dans lequel les critères de qualité sont associés avec un ensemble d'attributs de bas niveau directement mesurables qui sont normalement appelés *métriques de qualité*. Cette approche hiérarchique (Figure 3.2) a été suggérée en premier par McCall et al. (1977).

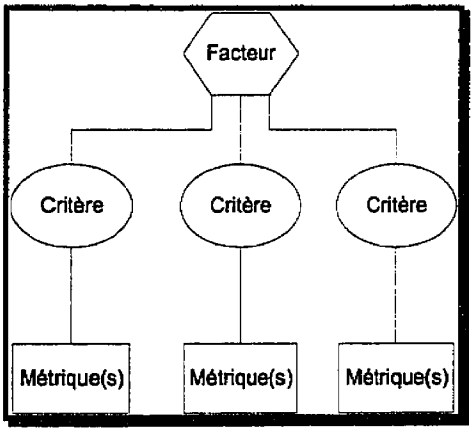


Figure 3.2 : architecture facteur-critère-métrique

Cavano et McCall (1978) stipulent que l'architecture de la Figure 3.2 contribue à mesurer quantitativement la qualité logicielle. De plus, toujours au dire de Cavano et McCall (1978), cette approche évite plusieurs embûches rencontrées par d'autres dans ce domaine au cours des dernières années. Un des avantages marqués est que cette approche n'essaie pas d'utiliser une seule mesure pour quantifier la qualité.

3.4.1 Problèmes liés aux modèles de qualité basés sur l'architecture facteur-critère-métrique

Kitchenham (1987) a dénoté un certain nombre de problèmes aux modèles de qualité qui utilisent cette architecture hiérarchique. Le plus important semble être la différence entre les facteurs de qualité identifiés par différents chercheurs et les critères sous-jacents associés aux facteurs sélectionnés. Un autre problème majeur est que l'architecture simple facteur-critère-métrique cache le fait que la nature des facteurs, critères et métriques est très différente pour des qualités différentes. Ainsi, les facteurs de qualité incluent ceux qui s'appliquent à des systèmes spécialisés, ceux qui s'appliquent d'une manière générale à toutes les classes de systèmes et ceux qui sont liés au processus de développement logiciel. La nature des critères sous-jacents est également très diverse : certains critères correspondent à une description plus détaillée du facteur de qualité, certains sont des caractéristiques fonctionnelles nécessaires pour assurer que le facteur de qualité soit supporté dans le système final et les autres sont des caractéristiques du processus de développement. Les métriques de qualité elles-mêmes sont un mélange de métriques authentiques (telles des mesures de structures des programmes), de listes de contrôle de procédures de design et de standards de production.

Les autres problèmes des modèles basés sur l'architecture facteur-critère-métrique dénotés par Kitchenham (1987) sont :

- il y a des chevauchements entre les différents facteurs de qualité (la Figure 3.2 qui a été tirée intégralement de Cavano et McCall (1978) ne reflète pas le fait qu'une métrique puisse être associée à plus d'un critère) ;
- il n'y a pas d'indications claires des compromis à faire en rapport aux relations entre les facteurs (par exemple le facteur *efficacité* contre celui de *maintenabilité*) ;
- il n'y a pas de relations explicites entre les critères, les métriques et le cycle de vie ;
- il n'y a aucun rationnel objectif pour inclure ou exclure un facteur de qualité en particulier ;
- les facteurs de qualité ne sont pas définis en termes mesurables. Ainsi, la validation des relations postulées entre les facteurs de qualité et les métriques de qualité est difficile.

D'autres problèmes ont été également dénotés par Kaposi et Kitchenham (1987) :

- les relations entre les facteurs et les critères de qualité sont des relations entre mots, chacun ayant une définition dépendante du contexte ;
- les critères sont mesurés en utilisant des métriques de qualité qui sont un mélange de métriques authentiques, d'opinions subjectives et d'attributs déterminés par analyse ou existence. Or, il semble qu'il n'y ait aucune reconnaissance explicite des différences entre ces types de métriques ;
- les métriques de qualité qui sont liées directement aux facteurs de qualité (tel le temps entre deux défaillances) sont exclues du modèle ;
- la nature subjective des relations entre les métriques de qualité et les critères de qualité jusqu'aux facteurs de qualité n'est pas reconnue et la nature subjective de l'évaluation globale de la qualité est ignorée.

3.5 Procédure d'élaboration d'un modèle de qualité logicielle

Dans cette section, nous présentons un aperçu des procédures d'élaboration connues d'un modèle de qualité (section 3.5.1). À partir de ces dernières, nous élaborerons une procédure qui semble être plus rigoureuse que celles développées jusqu'à présent (section 3.5.2). La section 3.6 quant à elle présente les aspects particuliers qu'il faut considérer lors de l'élaboration.

3.5.1 Procédures d'élaboration existantes

Selon Fenton (1991), un ingénieur logiciel et/ou un utilisateur logiciel peuvent utiliser le principe de la décomposition des attributs de qualité pour choisir ou développer un modèle de qualité :

- *Approche du modèle fixe* : ils acceptent que tous les facteurs de qualité importants dont ils ont besoin pour contrôler sont un sous-ensemble d'un modèle qui a été sélectionné. Pour contrôler et mesurer ces facteurs de qualité, ils acceptent les critères et métriques associés aux facteurs. Encore plus important, ils acceptent les relations entre les facteurs, les critères et les métriques.

- *Approche de définition de son propre modèle de qualité* : ils acceptent la philosophie générale de la décomposition des attributs de qualité. Cependant, l'ingénieur logiciel et l'utilisateur logiciel s'entendent sur les attributs de la qualité qui semblent les plus importants pour un certain produit (ou plutôt classe de produits). Ensuite, ils élaborent une décomposition (possiblement guidée par un modèle existant) dans laquelle ils sont d'accord sur les mesures spécifiques pour les attributs (critères) du plus bas niveau et les relations qui en découlent.

La deuxième approche de Fenton (1991) est basée sur l'approche descendante qui consiste à identifier premièrement un ensemble de facteurs de qualité et, d'une manière descendante, décomposer ces facteurs jusqu'à ce qu'on atteigne les caractéristiques du produit ou des métriques. Or, il est également possible de se baser sur l'approche ascendante. Dans cette approche, il faut commencer par identifier les caractéristiques du produit qui semblent être les plus intéressantes et ensuite procéder des attributs tangibles vers ceux qui sont intangibles en recherchant les facteurs de qualité qui sont sensibles aux caractéristiques du produit.

La majorité des modèles de qualité existants ont été développés selon l'approche descendante. C'est le cas, en particulier, des modèles de Boehm et al. (1978) et IEEE (1992).

Boehm et al. (1978) ont opté pour une approche itérative qui consiste à raffiner le modèle jusqu'à ce qu'il soit adéquat. L'approche est la suivante :

1. définir un ensemble de caractéristiques qui sont importantes pour le logiciel, qui n'est pas trop exhaustif et non chevauché ;
2. développer des métriques candidates pour évaluer le degré selon lequel le logiciel possède les caractéristiques ;
3. développer un ensemble de caractéristiques des métriques de qualité incluant des items tels les corrélations avec la qualité logicielle, le bénéfices potentiels, la quantifiabilité et la facilité d'automatisation ;
4. évaluer chaque métrique candidate selon les critères du point 3 et selon les interactions avec les autres métriques : chevauchements, dépendances, défauts, etc.;
5. en se basant sur les évaluations du point 4, raffiner l'ensemble des caractéristiques logicielles pour concevoir un ensemble davantage mutuellement exclusif et exhaustif selon les utilisations de l'évaluation de la qualité logicielle ;
6. raffiner les métriques candidates et les réaligner selon l'ensemble révisé des caractéristiques ;

7. développer des algorithmes, lorsque possible, pour les métriques révisées et évaluer le degré selon lequel les métriques captent l'essence de chacune des caractéristiques ;
8. retourner à l'étape 4. L'itération se poursuit jusqu'à ce que le modèle soit adéquat.

Il est à noter que l'étape 8 a été ajoutée parce qu'il n'est pas certain que nous réussirons à converger en un modèle de qualité adéquat en seulement deux itérations.

3.5.2 Procédure d'élaboration développée

Comme nous l'avons constaté à la section 3.5.1, les procédures d'élaboration existantes sont très succinctes et ne fournissent pas vraiment une façon rigoureuse d'élaborer un modèle de qualité. De plus, elles négligent deux aspects excessivement importants qui se doivent d'être considérés lors de l'élaboration, soit l'architecture du modèle et la procédure d'agrégation des données.

La procédure que nous proposons consiste en quatre étapes distinctes. Ces étapes sont présentées d'une façon sommaire ci-dessous, mais elles sont reprises en détails dans les quatre chapitres suivants.

1. **Élaboration de l'architecture du modèle de qualité (chapitre 4)**

   L'architecture correspond à la fondation d'un modèle de qualité et est par le fait même très importante puisqu'elle dicte le type de relations qui peuvent exister entre les différents attributs du modèle. Comme nous le verrons, il existe plusieurs architectures possibles, chacune ayant des forces et des faiblesses, ce qui complique grandement le choix de l'architecture qui convient le mieux aux besoins.

2. **Identification d'un ensemble d'attributs externes de qualité (chapitre 5)**

   Cette activité est critique puisqu'elle cristallise notre vision de la qualité. Or, comme il sera vu plus loin, il y a une multitude d'attributs externes de qualité qui existent et en choisir un ensemble qui est complet, compatible et non entrelacé s'avère une tâche très complexe.

3. **Identification d'un ensemble d'attributs internes de qualité (chapitre 6)**

   Ayant identifié les attributs de la qualité externes qui sont pertinents, il faut trouver une façon de mesurer leur degré d'atteinte. C'est précisément un des objectifs des métriques logicielles. Il s'agit donc dans cette étape de trouver un ensemble de métriques qui sont associées aux attributs externes de la qualité. Malheureusement, ce choix est très difficile puisqu'il apparaît que les métriques logicielles sont un des sujets les plus mal compris dans le domaine plus général du génie logiciel.

4. **Élaboration d'une procédure d'agrégation des données (chapitre 7)**

   Les trois premières étapes ont servi à concevoir le modèle de qualité en tant que tel. Or, le modèle seul n'est pas suffisant, il doit inclure une procédure pour agréger les valeurs des attributs de bas niveau jusqu'aux attributs de haut niveau. Encore une fois, le choix ou l'élaboration d'une procédure d'agrégation est une activité délicate puisqu'il faut qu'elle respecte la théorie de la mesure.

Bien que la procédure qui est proposée en soit une rigoureuse qui devrait permettre d'élaborer un modèle de qualité possédant tous les atouts désirables, il est impératif que le modèle résultant ne soit en aucun cas statique. Ainsi, il doit être constamment amélioré. Les améliorations peuvent prendre l'allure d'une nouvelle itération de la procédure d'élaboration. Ces améliorations s'inscrivent dans la lignée du concept de la qualité totale qui stipule entre autres que nous devons améliorer constamment la qualité des produits et des processus en vue de rester compétitif. Une autre raison qui nous

incite à faire évoluer le modèle est que le domaine de l'informatique est un domaine en pleine effervescence qui évolue rapidement.

3.6 Aspects à considérer lors de l'élaboration d'un modèle de qualité logicielle

Ce qui doit être reconnu lors de l'élaboration d'un modèle de qualité est qu'un logiciel ne manifeste pas directement des attributs de la qualité. Il expose plutôt des caractéristiques d'un produit qui impliquent ou contribuent à certains attributs de la qualité. Dans le même ordre d'idées, Pressant (1992) stipule que nous ne mesurons jamais vraiment la qualité mais plutôt une certaine manifestation de la qualité.

Selon Kaposi et Kitchenham (1987), comme dans n'importe quelle autre discipline, les modèles de qualité doivent présenter trois composantes : *concept*, *théorie* et expérience acquise par la *pratique*.

- Les *concepts* sont des idées clairement définies. Sans des fondations conceptuelles bien définies, les travailleurs dans le domaine ne peuvent pas communiquer efficacement. En l'absence d'un modèle conceptuel, il est difficile d'extraire des principes généraux comme des agrégats événements individuels et il est difficile pour une personne d'apprendre à partir de l'expérience d'un autre ;

- Les *théories* sont des systèmes d'idées. Elles sont des moyens pour raisonner de façon précise et faire des déductions fiables. Elles fournissent la base pour manipuler les concepts, c'est-à-dire tirer des conclusions à partir de ces derniers, les lier ensemble, prédire les résultats des événements et formuler des nouveaux concepts pouvant décrire les résultats prédits. Sans théories, un modèle conceptuel est une coquille statique. Les théories

créent la dynamique d'une discipline, elles amènent les concepts à la vie tout en leur accordant une utilité potentielle ;

- La *pratique* génère la substance qui remplit la coquille de la théorie. Sans faits, en l'absence de données acquises par l'entremise de la pratique, les théories peuvent avoir des mérites intellectuels sans utilités ; les concepts à l'intérieur d'une théorie peuvent être consistants les uns avec les autres, mais leur consistance avec la réalité est non prouvée. Dans une discipline pratique, telle l'ingénierie, les théories doivent avoir la pertinence pour résoudre des problèmes de la vie réelle. La validité des théories peut seulement être établie par la pratique.

L'évolution de n'importe quelle discipline d'ingénierie est le résultat d'une interaction constante entre les concepts maturants, les théories développées et les expériences pratiques accumulées. Le processus d'évolution est irrégulier : la formation des concepts peut suivre ou succéder l'accomplissement théorique, l'expérience pratique peut guider ou être guidée par la compréhension théorique. Dans le cours de cette évolution, le rôle des chercheurs change : la recherche peut être constructive, fournissant le stimulus pour des nouveaux développements, ou analytique expliquant rétrospectivement et étayant l'expérience pratique. Kaposi et Kitchenham (1987) sont d'avis que présentement la pratique et l'articulation de notions intuitives sont loin de la création de concepts et théories systématiques.

3.6.1 Mesures de processus et de produits

Un modèle de qualité a d'autres utilités que celle de servir de support pour l'évaluation finale de la qualité logicielle ; il peut également être utilisé pour prédire la qualité du produit final durant le développement et la maintenance logicielle (une discussion complète des utilités est présentée à la section 3.7). En extrapolant les dires de Banker

et al. (1991), il est reconnu que les modèles ne doivent pas séparer le développement logiciel de la maintenance logicielle.

Cette habileté à prédire la qualité peut servir à contrôler les processus de développement et de maintenance logicielle. Ainsi, pour être en mesure de répondre à ces dernières attentes, un modèle doit pouvoir gérer les métriques de processus et les métriques des produits intermédiaires.

Or, jusqu'à présent et selon Dromey (1995), les approches proposées intègrent seulement un des deux types d'indicateurs (ou de métriques) : ceux basés sur le processus ou ceux basés sur les produits. Toutefois, il semble que le modèle constructif de qualité (COQUAMO) (Kitchenham, 1987) soit passé inaperçu aux yeux de Dromey (1995) puisqu'il intègre à la fois des indicateurs de processus et de produits.

Les approches qui essaient d'évaluer ou de prédire la qualité des produits par l'entremise de métriques de produits ont été beaucoup critiquées. Ainsi, de tels efforts, qui fournissent de l'information seulement après que le produit ait été conçu, ont été caractérisés par Kearney et al. (1986), comme étant trop restreints et basés sur des métriques questionnables. À l'opposé, les approches qui s'intéressent au potentiel d'un processus à faire un produit de qualité, ne garantissent pas nécessairement que l'exécution d'un processus va résulter en un produit de qualité.

3.6.2 Automatisation de la collecte des données

Selon Arthur et al. (1993), les évaluations (mesures) reliées aux processus et aux produits doivent être en grande partie automatisées. Ces mesures, pour celles qui sont directes et objectives, sont pour la plupart faciles à obtenir. L'utilisation d'un tel outil, qui peut prendre la forme d'un analyseur statique, est justifiée par le fait qu'il y a beaucoup de mesures à prendre et qu'il serait pénible qu'elles soient prises

manuellement. Arthur et al. (1993) proposent un ensemble de règles qui devraient être respectées lors de la conception d'un tel outil.

3.6.3 Nombre de niveaux

Dromey (1995) stipule qu'élaborer un modèle qui présente un grand nombre de niveaux entre les facteurs de qualité et les caractéristiques du produit n'est pas la meilleure façon de procéder. Il opte pour un seul niveau (un ensemble de propriétés qui transportent certaines caractéristiques de la qualité) entre les facteurs de qualité et les caractéristiques du produit. ISO-9126 (1991) mentionne, que néanmoins, un modèle de qualité doit présenter au moins un niveau de sous-caractéristiques.

3.6.4 Validation

Selon Tichy et al. (1995), les publications tombant dans la catégorie du design ou de la modélisation nécessitent des expériences reproductibles pour valider leurs résultats. Ceci est imputable au fait que les résultats ne peuvent pas être prouvés par un raisonnement logique. Sans validation, les publications échouent à établir des résultats utiles et crédibles. Ce qui est en accord avec la composante *pratique* de Kaposi et Kitchenham (1987) (voir introduction de la section 3.6).

Fenton (1991) stipule qu'une expérimentation rigoureuse devrait être effectuée avant d'en arriver à des conclusions concernant les bénéfices résultant de l'utilisation d'une méthode (ou modèle de qualité pour ce qui nous concerne). Une telle expérimentation doit impliquer des projets réalistes d'envergure et non des projets artificiels reliés à des travaux d'étudiants. Évidemment, ce type d'expérimentation est long et coûteux ; c'est ce qui explique en partie qu'il y en ait si peu d'effectuées.

Toujours au dire de Fenton (1991), élaborer un design d'expérimentation n'est pas chose facile. En effet, il doit être cohérent avec les hypothèses sous observation. Une fausse représentation de la réalité causée par un mauvais choix d'attributs peut mener à conclure au bienfait d'une technique alors qu'une telle conclusion n'est pas fondée. Concevoir un design d'expérimentation adéquat ne garantit pas nécessairement une analyse sans faute. Ainsi, il faut que les mesures soient effectuées sur une quantité suffisante de données et que ces dernières soient pertinentes. Une autre complication provient du fait que les mesures doivent être manipulées adéquatement. Selon les données sous observation, certaines opérations mathématiques peuvent être effectuées tandis que d'autres sont prohibées (voir section 2.6).

3.6.5 Évaluation

Cavano et McCall (1978) présentent certaines caractéristiques (voir énumération ci-dessous) qui devraient être examinées pour évaluer un modèle de qualité. Bien que cette évaluation soit succincte, elle s'avère couvrir les aspects importants reliés aux modèles de qualité.

- *définition*
  - ⇒ qu'est-ce que le modèle mesure ?
  - ⇒ est-ce qu'il est suffisamment détaillé ?

- *fidélité*
  - ⇒ est-ce que différentes personnes du domaine de l'assurance qualité arriveront aux mêmes résultats ?
  - ⇒ est-ce que les faits sont suffisamment proches des prédictions ?

- *constructif*
  - ⇒ est-ce qu'il aide à comprendre la qualité logicielle ?
  - ⇒ est-ce que les mesures qui sont dérivées sont explicables ?

- *stabilité*
  - ⇒ est-ce que le modèle peut être manipulé pour obtenir les résultats désirés ?

- *utilisabilité*
  - ⇒ est-ce que la méthodologie peut être implantée efficacement (par rapport au coût) dans un programme d'assurance qualité ?

3.7 Utilités des modèles de qualité logicielle

Ayant en main un modèle de qualité, il est primordial au dire de Pfaltz (1977) d'en démontrer l'utilité. Il faut donc montrer que son utilisation permet d'obtenir des informations et des résultats pertinents sur le phénomène analysé, soit la qualité logicielle. Dans cet ordre d'idées, cette section présente les principales utilités de la majorité des modèles de qualité existants.

Nous avons regroupé les utilités dans quatre catégories qui sont présentées dans les quatre prochaines sous-sections (section 3.7.1 à 3.7.4). La sous-section 3.7.5 présente quant à elle l'utilité des modèles selon l'approche ascendante et descendante.

3.7.1 Établir les exigences de qualité et les critères d'acceptation pour un logiciel

Selon IEEE 1061-1992 (1992) et ISO-9126 (1991) les modèles de qualité peuvent être utilisés pour définir les exigences de qualité d'un logiciel dans la phase du cycle de vie

prévu à cet effet, soit lors de la définition des exigences ou des spécifications. Dans la même veine, Cavano et McCall (1978) mentionnent que les modèles fournissent un mécanisme pour les gestionnaires pour identifier quelles sont les qualités importantes. Ces qualités sont des attributs du logiciel différents de l'exactitude fonctionnelle et de la performance qui ont des implications au niveau du cycle de vie. De tels facteurs comme la *maintenabilité* et la *portabilité* ont un impact significatif sur le coût du cycle de vie. Le personnel d'assurance qualité logicielle reçoit par le fait une meilleure direction. Ainsi, il est rendu conscient des qualités qui sont considérées importantes et qui, par conséquent, devraient être vérifiées.

Les modèles de qualité fournissent également une structure rigoureuse pour définir les critères d'acceptation d'un projet. Ainsi, ces critères peuvent être spécifiés en terme de pourcentage d'atteinte de chacun des attributs de haut niveau (ex : *fiabilité* : 50%, *portabilité* : 10%, etc.). Cette utilisation est supportée par IEEE 1061-1992 (1992) qui ajoute que les modèles permettent d'établir des standards.

3.7.2 Évaluer et guider la progression du développement logiciel relativement aux objectifs (ou exigences) de qualité

L'objectif ultime des modèles de qualité est d'augmenter le contrôle sur les activités de développement et de maintenance en vue de concevoir des produits de meilleure qualité. Ainsi, selon IEEE 1061-1992 (1992), les modèles permettent d'atteindre les objectifs de qualité. Une des techniques permettant d'atteindre ces objectifs est de fournir, par l'entremise des modèles de qualité, des conseils sur la façon de s'assurer d'un certain niveau de qualité lors de la conception de logiciels.

Selon Cavano et McCall (1978), les modèles fournissent une façon pour évaluer quantitativement le niveau de progression du développement relativement aux objectifs de qualité établis. L'avantage des modèles sur les techniques courantes telles les inspections de code ou l'utilisation d'auditeurs du code source est la quantification

qu'ils introduisent. Ainsi, le personnel d'assurance qualité a un outil additionnel pour évaluer la qualité des produits intermédiaires. En fait, l'outil fournit une évaluation de la qualité dans une dimension différente (i.e., selon les facteurs de qualité qui sont liées à la vue des gestionnaires). Il y a une multitude de produits intermédiaires qui peuvent être évalués ; en voici quelques exemples :

- Les spécifications logicielles pour voir si elles satisferont les exigences de qualité durant le développement (ISO-9126, 1991) ;
- Le logiciel développé avant la livraison (ISO-9126, 1991) ;
- Le logiciel avant l'acceptation (ISO-9126, 1991).

Les évaluations de produits intermédiaires permettent de détecter, selon IEEE 1061-1992 (1992), des anomalies en cas de problèmes potentiels dans le système. Parallèlement IEEE 1061-1992 (1992) et Cavano et McCall (1978) stipulent que ces évaluations permettent de prédire le niveau de qualité qui sera atteint dans le futur avant les tests ou la livraison du système, ce qui permet d'identifier et de corriger plus rapidement les fautes et résultera en de grosses économies d'argent.

Du point de vue de IEEE 1061-1992 (1992), les modèles de qualité permettent en général :

- De suivre l'évolution de la qualité lorsque le logiciel est modifié ;
- D'évaluer la facilité de changement d'un système durant l'évolution du produit ;
- De normaliser, graduer, calibrer, ou valider des métriques.

En ce qui concerne le groupe d'assurance qualité au sein d'une organisation, Cavano et McCall (1978) mentionnent les deux points suivants :

- Les modèles augmentent l'interaction du personnel d'assurance qualité tout au long de l'effort de développement. Les métriques peuvent être établies pour que des sous-ensembles soient appliqués durant les phases du

développement. Le personnel d'assurance qualité ne vérifie pas seulement la spécification des exigences pour la conformité du format, mais il prend également des mesures qui peuvent identifier une faible qualité dans le document d'exigences qui peut éventuellement mener à un produit final déficient ;

- Des indications de faible qualité dans les premières phases peuvent pour une raison ou une autre ne pas être considérées immédiatement ; des actions correctives peuvent être retardées à cause d'activités plus prioritaires telles la livraison. Cependant, ils sont des indicateurs que des problèmes potentiels peuvent exister. Le personnel d'assurance qualité peut utiliser ces indications pour identifier de nouveaux standards qui devraient être suivis dans le futur, ou pour identifier des modules qui devraient recevoir une attention particulière lors des tests.

3.7.3 Évaluer si les exigences de qualité ou les critères d'acceptation d'un logiciel ont été rencontrés

La démarche pour vérifier si un logiciel répond aux exigences de qualité ou aux critères d'acceptation est simplifiée par l'entremise des modèles de qualité au dire de IEEE 1061-1992 (1992). D'une façon sommaire, il suffit de vérifier que l'évaluation de chacun des attributs de haut niveau soit supérieure ou égale aux critères d'acceptation.

3.7.4 Faciliter la communication des aspects de la qualité aux clients, utilisateurs et différents groupes de l'équipe de développement

De par leur architecture hiérarchique, les modèles de qualité facilitent la communication entre les divers intervenants. Ainsi, les clients, utilisateurs et

gestionnaires sont généralement intéressés à obtenir une vision globale de la qualité ; le niveau des facteurs de qualité répond à ce besoin. Les développeurs sont plus intéressés à des attributs plus tangibles qui leur permettent de mesurer le niveau de qualité du produit final mais aussi des produits intermédiaires ; les niveaux inférieurs des modèles de qualité répondent parfaitement à ces besoins.

D'après ISO-9126 (1991), les modèles de qualité peuvent servir à décrire les attributs de qualité d'un logiciel qui est implanté (i.e., dans un manuel d'usager). La description de ces attributs peut servir selon Boloix et Robillard (1994A) à classifier les projets logiciels selon le degré d'atteinte des attributs.

Une autre utilité des modèles de qualité est de fournir une structure pour interpréter les mesures de qualité soit au niveau du processus ou du produit.

3.7.5 Utilités des modèles de qualité selon l'approche ascendante et descendante

D'une manière descendante, selon IEEE std. 1061-1992 (1992), les modèles de qualité facilitent :

- L'établissement d'exigences de qualité en termes de facteurs par les gestionnaires tôt dans le cycle de vie du système ;
- La communication des facteurs établis en termes de sous-facteurs de qualité au personnel technique ;
- L'identification de métriques qui sont liées aux facteurs et sous-facteurs établis.

D'une manière ascendante, selon IEEE std. 1061-1992 (1992), les modèles de qualité permettent aux gestionnaires et au personnel technique d'obtenir des réactions en :

- Évaluant les produits et processus logiciel au niveau élémentaire des métriques ;
- Analysant les valeurs des métriques pour estimer et évaluer les facteurs de qualité.

3.8 Principaux intervenants

Lorsque le modèle est défini, il y a principalement deux façons de l'utiliser qui conviennent à deux groupes distincts de l'équipe de développement. Premièrement, il peut être utilisé selon une perspective ascendante. Selon Dromey (1995), cette perspective est plus utile pour ceux qui ont la responsabilité d'implanter des logiciels de qualité, c'est-à-dire les programmeurs. Deuxièmement, on retrouve la perspective descendante qui est l'inverse de la première. Encore une fois, selon Dromey (1995), cette perspective est plus utile aux designers qui ont la responsabilité de spécifier et d'implanter les attributs de haut niveau de la qualité dans le design du logiciel.

Dans la même veine, IEEE 1061-1992 (1992) stipule que les modèles de qualité peuvent être utilisés par :

- Un gestionnaire de projet pour identifier, définir et prioriser les exigences de qualité pour un système ;
- Un développeur de système pour identifier des traits spécifiques qui devraient être incorporés dans le logiciel pour répondre aux exigences de qualité ;
- Une organisation d'assurance/contrôle/audition de la qualité et un développeur de système pour évaluer si les exigences de qualité ont été rencontrées ;
- Une personne chargée de la maintenance pour assister lors des changements de gestion durant l'évolution du produit ;

- Un utilisateur pour assister lors de la spécification des exigences de qualité pour un système.

Cavano et McCall (1978) stipulent que l'élaboration et l'évolution d'un modèle de qualité devraient être une nouvelle fonction du groupe d'assurance qualité au sein d'une organisation.

CHAPITRE IV
ARCHITECTURE DES MODÈLES DE QUALITÉ LOGICIELLE

La première étape à réaliser lors de l'élaboration d'un modèle de qualité est de choisir l'architecture qui semble la plus appropriée. Malheureusement, il semble que cette activité ait été souvent négligée puisque plusieurs des modèles de qualité existants sont basés sur des architectures plutôt douteuses.

Ce chapitre a pour objectif de présenter les principaux aspects des architectures existantes. Nous expliquons en premier lieu ce qu'est une architecture (section 4.1) et nous présentons ensuite quelles sont les principales qualités recherchées d'une architecture (section 4.2). Par la suite, nous élaborons une classification des architectures possibles, classification qui est basée sur la théorie des graphes (section 4.4). Nous fournissons également une analyse de chacune des architectures contenues dans la classification (section 4.5). Mais, avant d'entrer dans le coeur de la classification et de l'analyse, nous présentons quelques définitions et concepts de la théorie des graphes (section 4.3).

4.1 En quoi consiste l'architecture d'un modèle de qualité logicielle

L'architecture s'avère être la fondation d'un modèle de qualité et est par le fait même excessivement importante puisqu'elle dicte le type de relations qui peuvent exister entre les attributs qui composent le modèle. L'architecture (ou la structure) fait abstraction de l'aspect syntaxique des éléments qui la composent; de ce fait elle ne considère pas la définition ou le choix des attributs de qualité, mais seulement le type de relations qui peuvent exister entre ces attributs.

L'architecture et la méthode d'agrégation des données sont deux composantes d'un modèle de qualité qui sont intimement liées. La méthode d'agrégation est basée sur l'architecture du modèle; en d'autres termes c'est l'architecture qui dicte quel type d'agrégation est possible. Il faut donc s'assurer que l'architecture et la méthode d'agrégation sont compatibles et permettent d'obtenir le type d'évaluation de la qualité désirée.

4.2 Qualités désirables d'une architecture

Au dire de Dromey (1995), le défi, lorsque nous proposons un modèle de qualité logicielle, est de trouver une architecture qui peut accommoder toute la connaissance relative à la qualité logicielle et cela, d'une façon constructive, raffinable et intellectuellement gérable. Ainsi, l'architecture doit être compréhensible à plusieurs niveaux, décomposable et adaptable.

4.3 Définitions et concepts de la théorie des graphes

La classification des architectures est basée sur la théorie des graphes. Le langage des graphes est un langage puissant permettant de représenter simplement la structure d'un grand nombre de situations. Cette section présente les définitions et concepts généraux de la théorie des graphes qui sont pertinents à notre étude. Ces définitions et concepts proviennent de Gondran et Minoux (1985), Berge (1970) et Tremblay et Sorenson (1984).

Un graphe G consiste en un ensemble non vide V appelé l'ensemble de *sommets* du graphe, un ensemble E qui est l'ensemble des *arcs* du graphe et une correspondance entre l'ensemble des arcs E à un ensemble de paires d'éléments de V. Nous

considérons que les ensembles E et V d'un graphe sont des ensembles finis. Il est commode d'utiliser la notation $G=(V,E)$ lorsqu'on réfère à un graphe.

Si un arc $x \varepsilon E$ est associé avec une paire de sommets (u,v) où $u, v \varepsilon V$, alors nous disons que l'arc x connecte ou joint les sommets u et v. N'importe quels deux sommets qui sont connectés par un arc sont appelés des sommets *adjacents*. Si une des extrémités d'un arc $x \varepsilon E$ touche un sommet $u \varepsilon V$, nous disons que x est *incident* à u. Dans un graphe, un sommet qui n'est adjacent à aucun autre sommet est appelé un *sommet isolé*. Un graphe contenant seulement des sommets isolés est un *graphe nul*.

Dans notre cas, nous considérons seulement les *graphes non orientés* puisque les arcs n'ont pas d'orientation spécifique pour notre application. Également, tous les graphes considérés sont des *graphes simples*, il ne peut donc pas y avoir plus d'un arc entre deux sommets et les boucles ne sont pas permises (arc dont les deux extrémités pointent au même sommet).

Certains des graphes que nous utiliserons sont des *graphes connexes*, c'est-à-dire que pour toute paire (x,y) de deux sommets distincts, il existe une *chaîne* reliant les deux sommets. Une chaîne de longueur $q>0$ est une séquence $u=(u_1,u_2,...,u_q)$ d'arcs de G telle que chaque arc de la séquence a une extrémité en commun avec l'arc précédent et l'autre extrémité en commun avec l'arc suivant. Puisque nous considérons seulement les graphes non orientés, une chaîne est équivalente à un *chemin* en ce qui nous concerne. Un *graphe non connexe* n'a donc pas de chaîne reliant chaque paire de sommets distincts.

D'autres types de graphes qui sont utiles sont les *graphes bipartis*. Un graphe est biparti si l'ensemble de ses sommets peut être partitionné en deux classes X_1 et X_2 de sorte que deux sommets de la même classe ne soient jamais adjacents. Il se note parfois $G=(X_1,X_2,U)$. Nous utilisons également une généralisation des graphes bipartis soit les *graphes N-partis* qui peuvent être notés $G=(X_1,X_2,..,X_n,U)$.

Un graphe qui ne contient pas de *cycle* est appelé un *graphe acyclique*. Un cycle est une chaîne $u=(u_1,u_2,...,u_q)$ telle que : le même arc ne figure pas deux fois dans la séquence et les deux sommets aux extrémités de la chaîne coïncident. Une famille importante de *graphes acycliques connexes* sont les *arbres*. Les arbres que nous considérons sont des arbres où l'ordre relatif des noeuds à un niveau quelconque n'est pas important ; il s'agit donc d'*arbres non ordonnés*. Un graphe acyclique qui n'est pas connexe est appelé une *forêt* (chaque composante connexe de la forêt est un arbre). Il résulte de la définition qu'un arbre est toujours un graphe simple.

Dans un arbre, un sommet qui n'a pas d'enfant est appelé un sommet terminal ou une *feuille* et tous les autres sommets sont appelés des *noeuds*.

Plusieurs des arbres que nous utilisons sont appelés des *arborescences* qui sont des arbres munis d'une *racine*. Une racine est un point a tel que tout autre sommet du graphe puisse être atteint par un chemin issu de a. La condition nécessaire et suffisante pour qu'un graphe $G=(X,U)$ admette une racine est qu'il soit *quasi-fortement connexe*. Or, un graphe est dit quasi-fortement connexe si pour tout couple de sommets (i,j), il existe un sommet k relié à i et à j par deux chemins, l'un entre k et i, l'autre entre k et j.

Dans une arborescence, le niveau d'un sommet correspond à la longueur du chemin de la racine à ce sommet. Le niveau de la racine est évidemment zéro.

Une autre classe d'arbres d'intérêt sont les arbres *M-aires*. Ces arbres ont la caractéristique que chaque noeud est au maximum de *degré M*. Le degré d'un noeud correspond aux nombres d'arcs ayant une extrémité qui touche le noeud. Si le degré de chaque noeud est exactement M et que le nombre de noeuds au niveau i est m^i alors l'arbre est appelé un *arbre M-aires complet*. Pour $M=2$, ces arbres sont appelés des *arbres binaires* et des *arbres binaires complets*. Les arbres binaires sont très utilisés en informatique. Il est possible de représenter n'importe quel arbre sous la forme d'un arbre binaire. Évidemment, le concept des arbres M-aires et M-aires complets peut être porté aux arborescences qui peuvent être des arborescences M-aires et M-aires complètes.

La définition d'un graphe ne contient aucune référence à la longueur, la forme et la position des arcs joignant n'importe quelle paire de sommets et ne décrit aucun ordonnancement de la position des sommets. Par conséquent, pour un certain graphe, il n'y a pas de diagramme unique qui le représente. Or, nous essayerons, dans le mesure du possible, de représenter les graphes par une structure de type hiérarchique, structure qui a comme propriété d'étaler les sommets sur plusieurs niveaux.

4.4 Classification des architectures

Dans cette section, nous développons une classification des architectures qui prend racine dans les définitions et concepts décrits à la section 4.3. Il est à noter que le terme architecture correspond non pas à une seule architecture mais plutôt à une famille d'architectures qui possèdent toutes les mêmes propriétés. Or, nous utiliserons quand même le terme architecture pour alléger la description.

La classification initiale que nous avions développée comportait plus d'une centaine d'architectures. Or, après étude de cette classification, il est apparu que beaucoup d'entre elles n'étaient pas pertinentes pour agir à titre de structure d'un modèle de qualité logicielle. Ainsi, dans la classification qui est présentée à la Figure 4.1, nous avons éliminé toutes les architectures non pertinentes et gardé seulement celles qui présentaient un intérêt pour notre domaine d'étude. Ironiquement, nous utilisons un *arbre* pour présenter la classification. Dans cet arbre de classification, les sommets correspondent à des architectures. La racine de l'arbre de classification correspond à une architecture très générale et, au fur et à mesure que nous parcourons l'arbre en profondeur, les noeuds rencontrés correspondent à des architectures de plus en plus spécifiques. Ainsi, une architecture du niveau i est un sous-ensemble de l'architecture du niveau i-1. La Figure 4.2 fournit une représentation typique pour chacune des architectures de la classification.

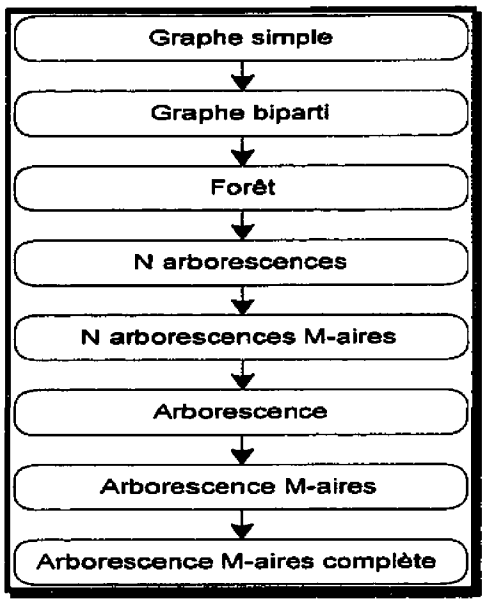


Figure 4.1: classification des architectures

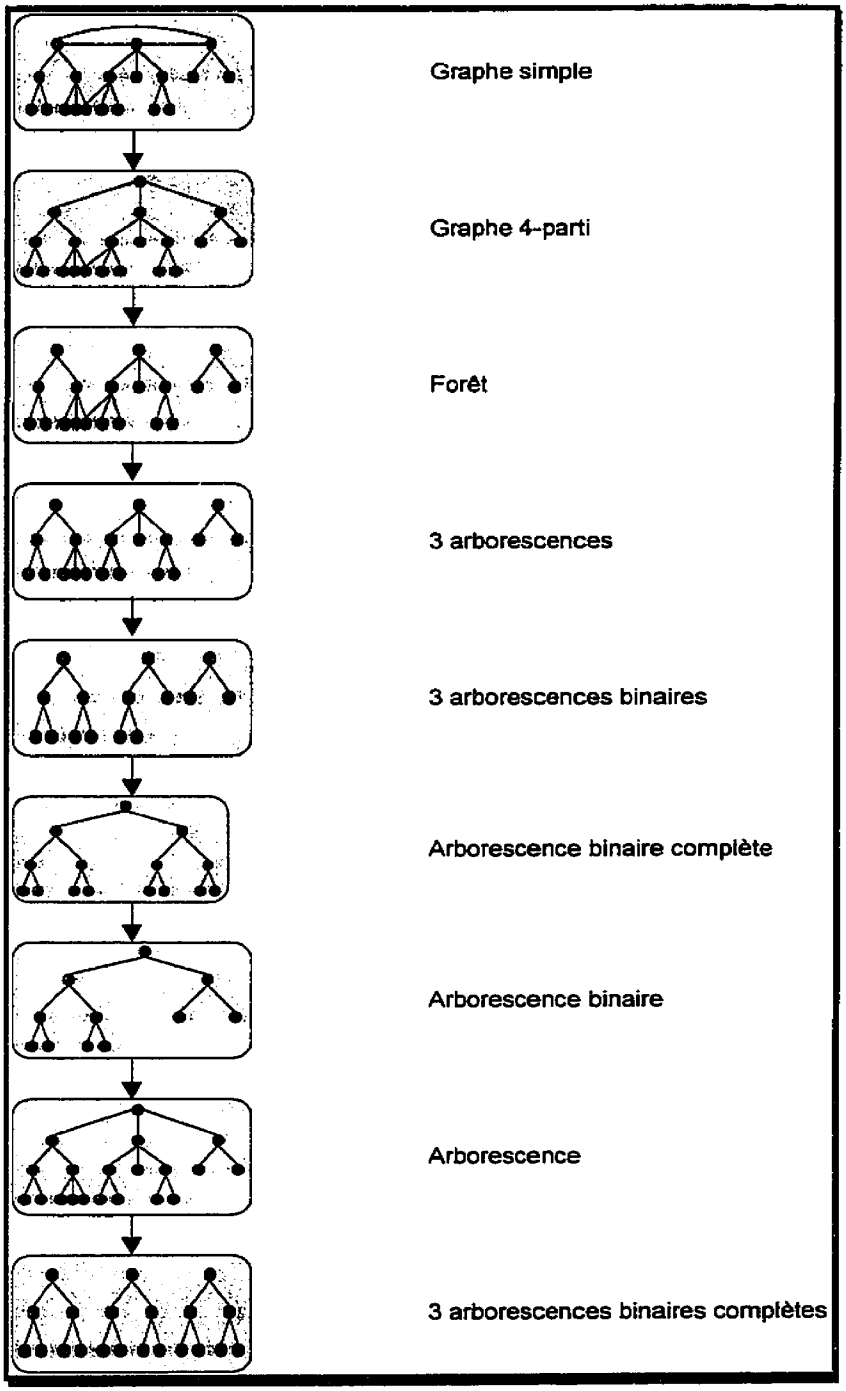


Figure 4.2: exemple typique pour chacune des architectures

Il s'avère intéressant d'appliquer la classification que nous venons de développer aux modèles de qualité logicielle existants, ce qui est l'objet du Tableau 4.1.

Tableau 4.1 : classification des modèles de qualité logicielle selon leur architecture

Architecture	Modèle de qualité logicielle
Graphe simple	
Graphe nul	ISO-9126 (1991)
Graphe N-parti	Boehm et al. (1978) Arthur et al. (1993)
Forêt	Dromey (1995) McCall et al. (1977)
N arborescences	Grady et Caswell (1987) IEEE1061-1992 (1992) ISO-9126 (1991)
N arborescences M-aires	
N arborescences M-aires complètes	Boloix et Robillard (1994A)
Arborescence	
Arborescence M-aires	
Arborescence M-aires complète	

4.5 Analyse des architectures

Dans cette section, nous analysons chaque architecture selon plusieurs critères. Les critères ont été choisis en vue d'évaluer le degré selon lequel un type d'architecture est raffinable et intellectuellement gérable.

Les critères sont décrits ci-dessous :

- *Ajout d'un sommet*

  Évaluation de la facilité d'ajouter un nouveau sommet à l'architecture. Typiquement, il s'agirait de l'apparition d'un nouvel attribut de qualité ou de la reconsidération d'un attribut qui avait été précédemment négligé.

- *Ajout d'un arc*

  Évaluation de la facilité d'ajouter un arc à l'architecture. Un arc peut être ajouté si une nouvelle relation entre deux attributs de qualité est découverte ou si une relation négligée est reconsidérée.

- *Ajout d'un sommet et d'un ou de plusieurs arcs*

  Évaluation de la facilité d'ajouter un nouveau sommet avec l'arc ou les arcs qui joignent le sommet à l'architecture. Comme pour l'ajout d'un sommet, il s'agirait de l'apparition d'un nouvel attribut de qualité ou de la reconsidération d'un attribut qui avait été précédemment négligé mais cette fois-ci avec le lien ou les liens qui unissent cet attribut avec un ou plusieurs des autres attributs.

- *Suppression d'un sommet*

  Évaluation de la facilité d'enlever un sommet à l'architecture. Cette situation survient lorsqu'un attribut de qualité devient désuet ou qu'il s'avère non pertinent pour un certain type d'application. Il est à noter que la suppression d'un sommet affecte seulement les sommets isolés pour que la suppression ait un sens, i.e. pour qu'il n'y ait pas d'arc en suspens.

- *Suppression d'un arc*

  Évaluation de la facilité d'enlever un arc à l'architecture. Une relation entre deux attributs de qualité peut s'avérer non fondée et par le fait même doit être supprimée.

- *Suppression d'un sommet et d'un ou de plusieurs arcs*

  Évaluation de la facilité d'enlever un sommet avec l'arc ou les arcs qui lui sont incidents. Comme pour la suppression d'un sommet, un attribut de qualité peut devenir désuet ou non pertinent pour un certain type d'application mais, dans ce cas-ci, en plus de supprimer l'attribut, il faut également supprimer le ou les liens qui lui sont incidents.

- *Indépendance des sommets d'un même niveau*

  Évaluation du degré de chevauchement ou d'entrelacement possible entre les sommets d'un même niveau. Les attributs de qualité sont non chevauchés s'il n'y a pas de liens entre les attributs d'un même niveau et que les attributs d'un même niveau ne dépendent pas des mêmes sous-attributs.

Le Tableau 4.2 présente les résultats de l'analyse. Dans ce tableau, une lettre est utilisée pour indiquer les contraintes qui doivent être respectées lors de l'ajout ou de la suppression d'un sommet et/ou un ou des arcs. La signification de chacune des lettres est décrite dans l'énumération qui suit. Une lettre *i* quelconque implique toutes les autres lettres qui sont avant elle dans l'alphabet (par exemple, la lettre C implique que les conditions A, B et C doivent être remplies). Il est à noter que les contraintes pourraient être affaiblies en tenant compte seulement des propriétés des graphes, mais nous avons préféré imposer des contraintes plus strictes pour prendre en considération les aspects particuliers de notre domaine d'étude. Par exemple, pour un graphe simple, la suppression de n'importe quel arc n'affecte pas ses propriétés, le graphe peut encore être considéré comme un graphe simple. Or, nous avons imposé une contrainte à la suppression d'un arc dans un graphe simple qui est que l'arc supprimé ne doit pas créer

de sommet isolé. L'indication N/A dans le tableau signifie que le critère est non applicable ou que son application ne fait pas de sens.

A- Aucune contrainte ;
B- L'arc ou les arcs ajoutés ne doivent pas être des boucles et ils ne doivent pas joindre deux noeuds qui sont déjà joints ;
C- Un arc qui est supprimé ne doit pas créer de sommet isolé alors qu'un sommet qui est supprimé doit obligatoirement être un sommet isolé ;
D- L'arc ou les arcs ajoutés peuvent joindre seulement deux sommets qui sont dans des classes adjacentes ;
E- Il ne doit pas y avoir de cycle créé;
F- Il doit exister un chemin entre la racine et tous les autres sommets de l'arborescence ou des N arborescences ;
G- Ce sont seulement les feuilles de l'arborescence ou une des N arborescences qui peuvent être supprimées avec l'arc qui leur est incident ;
H- Chaque noeud ne doit pas avoir plus de M descendants ;
I- Il faut enlever ou ajouter un niveau complet à l'arborescence ou à chacune des N arborescences.

Tableau 4.2 : analyse des architectures

Graphe simple	A	B	B	C	C	C	Non
Graphe nul	A	N/A	N/A	A	N/A	N/A	Oui
Graphe N-parti	A	D	D	C	C	C	Non
Forêt	A	E	E	C	C	C	Non
N arborescences	N/A	N/A	F	N/A	N/A	F	Oui

N arborescences M-aires	N/A	N/A	H	N/A	N/A	G	Oui
N arborescences M-aires complètes	N/A	N/A	I	N/A	N/A	I	Oui
Arborescence	N/A	N/A	F	N/A	N/A	G	Oui
Arborescence M-aires	N/A	N/A	H	N/A	N/A	G	Oui
Arborescence M-aires complète	N/A	N/A	I	N/A	N/A	I	Oui

Certaines conclusions peuvent être tirées à partir du tableau précédent :

- Les arborescences étant des graphes imposant une certaine structure, il n'est pas possible de manipuler un sommet sans considérer le ou les arcs qui lui sont incidents. Conséquemment, il n'est également pas possible de manipuler un arc sans considérer les sommets auxquels il est incident;

- Plus une architecture impose une structure stricte, plus il est contraignant de manipuler les éléments qui la composent. Les structures les plus contraignantes sont les arborescences M-aires complètes.

L'analyse de cette section sera d'une aide précieuse pour le choix de l'architecture lors de l'élaboration de notre modèle de qualité logicielle.

CHAPITRE V
ATTRIBUTS EXTERNES : LES FACTEURS ET CRITÈRES DE QUALITÉ

Ce chapitre a pour objectif de présenter les aspects généraux à considérer lors du choix des attributs externes d'un modèle de qualité logicielle. Nous débutons par une discussion sur la distinction entre un facteur et un critère de qualité (section 5.1). Nous présentons ensuite les exigences liées aux choix de ces attributs (section 5.2), suivi des difficultés liées à ces choix (section 5.3).

Nous tenterons d'adapter le cadre de Garvin (1984) qui identifie les huit dimensions essentielles à la qualité au domaine plus spécifique du logiciel (section 5.4). Également, nous présentons une classification des attributs externes de qualité selon la perspective de Kitchenham (1987) (section 5.5) ainsi qu'une classification que nous avons développée à partir d'une dizaine de modèles de qualité logicielle (section 5.6).

5.1 Facteur de qualité vs critère de qualité

Normalement un facteur de qualité représente une vision de la qualité selon un niveau d'abstraction plus élevé qu'un critère de qualité. Les critères servent donc à définir d'une façon plus précise les facteurs.

Or, malheureusement, la frontière entre un facteur et un critère de qualité est floue dans la littérature. Ainsi, il arrive qu'un facteur d'un modèle corresponde à un critère d'un autre modèle, l'inverse étant également vrai.

Notre but dans ce chapitre n'est pas de jouer l'avocat du diable en déterminant quels sont les attributs externes pouvant être considérés comme des facteurs et quels sont

ceux qui correspondraient plus à des critères, mais plutôt de présenter les principaux aspects de ces attributs. C'est pourquoi nous ne tiendrons pas compte de la nuance entre un facteur et un critère de qualité.

5.2 Exigences liées aux attributs externes de qualité

Traditionnellement, la *fiabilité* était le seul indicateur de qualité. Or, maintenant tous s'entendent pour dire qu'il n'existe pas un attribut unique qui peut prétendre mesurer toutes les facettes de la qualité logicielle. Ainsi, sachant que nous devons identifier non pas un seul attribut, mais plutôt un ensemble d'attributs, l'idéal serait, selon Dromey (1995), que cet ensemble soit complet, compatible et non entrelacé.

Lors de l'identification et de la définition de ces attributs, l'utilisateur et l'utilisation de ces attributs doivent être considérés. L'utilisateur est le gestionnaire, soit le client des développeurs du système logiciel. L'utilisateur a besoin d'un ensemble défini d'attributs pour identifier quelles sont les qualités désirées dans le produit logiciel développé. Pour satisfaire cette utilisation, les définitions des attributs doivent mener à une quantification qui a un sens pour l'utilisateur.

Les exigences pour choisir les caractéristiques de qualité du standard ISO-9126 (1991) furent les suivantes :

- couvrir tous les aspects de la qualité logicielle résultant de la définition de la qualité par ISO8402 ;
- décrire la qualité des produits avec un minimum d'entrelacement ;
- être le plus près possible de la terminologie établie ;
- former un ensemble de six à huit caractéristiques pour des raisons de clarté et de manipulation ;
- identifier les domaines d'attributs des produits logiciels pour des raffinements ultérieurs.

Les exigences de ISO-9126, bien qu'étant assez simplistes, semblent couvrir tous les aspects essentiels pour que l'ensemble des attributs de qualité choisi soit cohérent. Malheureusement, comme nous le verrons à la section 5.3, il est excessivement difficile de répondre parfaitement à toutes ces exigences.

5.3 Difficultés liées au choix des attributs externes de qualité

La première difficulté liée au choix des attributs externes de qualité est que nous ne sommes pas à ce jour capables de nous entendre précisément sur la définition et la portée de chacun des attributs. Une analyse semblable à celle faite à la section 1.4 sur la définition de la qualité logicielle pourrait être réalisée pour chacun des attributs de qualité.

Comme Dromey (1995) et ISO-9126 (1991) le mentionnent, un ensemble d'attributs externes de qualité devrait être complet, compatible et non entrelacé. Malheureusement, cette tâche s'avère très difficile pour ce qui concerne le logiciel. Ainsi, chaque attribut externe dépend d'un certain nombre d'attributs internes qui ne sont certainement pas mutuellement exclusifs dans leurs impacts sur la qualité de haut niveau. Nous avons déjà analysé le chevauchement des attributs au chapitre 4. Cette difficulté à trouver des attributs de qualité non chevauchés est partiellement expliquée par la maxime du naturaliste John Muir : « when we try to pick up out anything by itself, we find it connect to the entire universe ». Encore selon Dromey (1995), il n'y a pas grand chose que l'on puisse faire pour contrer ce problème. Il faut cependant s'assurer que les liens entre les attributs soient clairement établis.

Kitchenham (1987) illustre très bien la difficulté à choisir les attributs de qualité pour un domaine particulier du logiciel : en prenant certains attributs qui se retrouvent souvent dans les modèles de qualité, on s'aperçoit qu'il y a souvent un certain degré de

chevauchement entre les attributs liés à la rectification ou l'amélioration de logiciels existants.

Dans cet ordre d'idées, les attributs suivants sont analysés :

- *portabilité* : l'effort requis pour transférer un programme d'une configuration matérielle et/ou d'un environnement logiciel à un autre ;
- *interfonctionnement* : l'effort requis pour lier un système à un autre ;
- *réutilisabilité* : le degré selon lequel un programme peut être utilisé dans d'autres applications ;
- *maintenabilité* : la facilité selon laquelle les fautes et leurs erreurs associées peuvent être respectivement identifiées et corrigées ;
- *extensibilité* : l'effort pour incorporer des caractéristiques nouvelles ou étendues dans un système logiciel.

Les attributs *portabilité* et *interfonctionnement* semblent être intimement liés avec l'attribut *extensibilité* puisqu'ils sont concernés pour fournir des caractéristiques additionnelles à un système existant. La différence est peut être incorporée dans l'idée de *généralité* sans laquelle la *portabilité* ou l'*interfonctionnement* peuvent être impossibles. Dans ce sens, l'attribut *généralité* réfère aux caractéristiques qui existent pour permettre à un système d'être utile dans d'autres environnements (par exemple l'utilisation de langage de programmation de haut niveau ou de techniques telles le masquage d'informations).

Pour ce qui est de l'attribut *maintenabilité*, il semble y avoir deux éléments distincts : un concerné par la facilité à diagnostiquer les fautes et un concerné par l'effort impliqué dans la correction des erreurs. Le dernier élément de l'attribut *maintenabilité* chevauche fortement avec l'attribut *extensibilité* dans le sens que la correction est une extension à une caractéristique existante pour atteindre le niveau de fonctionnalité ou de performance désiré.

Les attributs *réutilisabilité* et *extensibilité* semblent être à l'opposé d'un spectre subjectif. Lorsqu'un programme se voit ajouter de nouvelles caractéristiques, est-ce que l'ancien système est étendu pour incorporer des nouvelles caractéristiques ou l'ancien système est-il réutilisé pour produire un nouveau système ? Il semble toutefois que l'attribut *réutilisabilité* soit un concept qui est plus approprié aux composantes d'un système (par exemple : un module ou une procédure) qu'à un système.

Grâce à la dernière discussion tirée de Kitchenham (1987), on s'aperçoit que choisir les attributs de qualité est une activité ardue et critique qui se doit d'être effectuée avec beaucoup de rigueur. En fait, échouer à trouver un ensemble d'attributs qui s'avère être complet, compatible et non entrelacé peut avoir comme conséquence le discrédit du modèle.

5.4 Les huit dimensions de la qualité de Garvin

Garvin (1984) a identifié huit dimensions qui servent de cadre pour raisonner sur les éléments à la base de la qualité des produits. Ces dimensions sont : la *performance*, les *particularités*, la *fiabilité*, la *conformité*, la *durabilité*, le *niveau de service*, *l'esthétique* et le *niveau de qualité perçu*. Les définitions de chacune de ces dimensions sont données à la fin de la présente section.

Ces huit dimensions majeures de la qualité couvrent un grand nombre de concepts. Plusieurs des dimensions impliquent des attributs de produits mesurables ; d'autres reflètent des préférences individuelles. Certaines sont objectives et sans notion de temps alors que d'autres évoluent selon les tendances. Certaines sont des caractéristiques inhérentes des biens, alors que d'autres sont des caractéristiques imputées aux biens.

Au dire de Garvin (1984), la diversité de ces concepts permet d'expliquer les différences entre les cinq approches génériques de la qualité (voir section 1.3).

Chacune de ces approches se concentre implicitement sur des dimensions différentes de la qualité : l'approche basée sur le produit se concentre sur la performance, les caractéristiques et la durabilité ; l'approche basée sur le client se concentre sur l'esthétique et le niveau de qualité perçu ; et l'approche basée sur le processus se concentre sur la conformité et la fiabilité. Les conflits entre ces approches sont inévitables puisque chacune définit la qualité d'un point de vue différent. Cependant, une fois les concepts séparés et chaque dimension considérée séparément, les sources de désaccord deviennent claires.

Gilb (1984) prétend que chacune des dimensions est complète et distincte, c'est-à-dire qu'un produit peut être classé élevé dans une dimension tout en étant faible dans une autre dimension. Selon nous, bien que cette affirmation soit hautement souhaitable, il semble cependant qu'elle soit gratuite et sans véritable fondement.

Une analyse faite par Kitchenham et Walker (1986) visant à faire un rapprochement entre les huit dimensions de Garvin (1984) et les attributs de qualité logicielle est présentée ci-dessous. Cette analyse est excessivement pertinente, mais malheureusement elle se limite à l'étude d'un ensemble d'attributs de qualité logicielle qui est plutôt restreint. La définition de chacune des dimensions selon Garvin (1984) est présentée en même temps que les résultats de l'analyse.

- la **performance** a des aspects internes et externes. Premièrement, elle réfère aux caractéristiques d'opérations principales d'un produit, mais ces dernières font partie de la spécification du produit, donc elles ne sont pas séparables en un ou des attributs de qualité en tant que tel. Cependant, la performance inclut l'*efficacité* selon laquelle le produit exécute ses fonctions. Cet aspect est un attribut de qualité distinguable ;

- les **particularités** sont définies comme étant des caractéristiques secondaires qui complètent le fonctionnement de base d'un produit. Néanmoins, elles font partie de la spécification d'un produit, donc elles ne forment pas un attribut de qualité séparé. Les attributs de qualité logicielle pertinents à cette

dimension de la qualité incluent l'*intégrité* et la *faculté de survie* qui sont des particularités de systèmes logiciels spéciaux et la *généralité* qui est une caractéristique du design requise pour des systèmes portables et interfonctionnables ;

- la **fiabilité** est la probabilité de défaillance d'un produit dans un temps donné et est aussi un attribut de qualité autant pour le logiciel que pour n'importe quel autre type de produits ;

- La **conformité** aux spécifications est la clé de l'approche basée sur le processus de la qualité (voir section 1.3). À l'intérieur d'une organisation, les déficiences au niveau de la conformité sont souvent mesurées en termes des défauts découverts durant le développement et la production. Subséquemment, elles apparaissent surtout sous forme de rapports de déficiences de la part du client. De la façon dont la conformité ou plutôt sa déficience se manifeste, elle est très liée à l'attribut de qualité *fiabilité*. D'autres qualités logicielles reliées sont la *justesse* et les qualités au niveau du design qui sont supposées mener à la conformité telle la *testabilité* et la *compréhensibilité* ;

- la **durabilité** peut être définie comme le nombre d'utilisations d'un produit avant qu'il ne soit physiquement détérioré. L'application de ce concept au logiciel nous mène à plusieurs considérations diverses, ce qui fait ressortir plusieurs attributs de qualité. La détérioration physique, en tant que telle, arrive seulement si, pour une période de temps, le logiciel est modifié d'une façon à le rendre incorrect ou non maintenable. Les attributs pertinents sont donc la *maintenabilité* et l'*extensibilité* où le dernier inclut l'habileté du logiciel à être modifié ou amélioré. Dans un contexte logiciel, la durabilité implique également la notion de persistance du design ou plus précisément des modules, ce qui nous amène à l'attribut *réutilisabilité*. De plus, il y a l'habileté du produit à survivre sous des conditions défavorables, ce qui peut être défini comme la *faculté de survie* ;

- le **niveau de service** concerne la rapidité, la courtoisie et la compétence des réparateurs et est essentiellement un synonyme pour l'attribut *maintenabilité* ;

- **l'esthétique** et le **niveau de qualité perçu** sont les dernières dimensions et elles sont hautement subjectives. Ainsi, elles sont associées à la vue transcendantale de la qualité. L'esthétique est dépendante du jugement personnel des clients et est basée sur l'apparence du produit, sa sensation et son goût. Le niveau de qualité perçu est relié à l'esthétique, mais il est plus général puisqu'il concerne la vue globale subjective du produit. L'attribut de qualité qui se rapproche le plus de ces deux dimensions est l'*utilisabilité* dans le sens de la facilité d'utilisation d'un système et le sujet relié de l'acceptance du système de la part des clients ;

D'après Kitchenham et Walker (1986), même si Garvin (1984) a identifié l'approche basée sur la valeur comme une de ses vues de la qualité, il n'identifie pas de dimension de la qualité directement reliée au coût. De tous les chercheurs en qualité logicielle, il semble que seulement Gilb (1986) considère le coût ou l'utilisation des ressources explicitement lors de la spécification de la qualité. Cependant, si la qualité est pour être suivie et évaluée durant le processus de conception, il semble clair que l'utilisation des ressources fournit de l'information utile sur la progression du processus et permet aussi de supporter la vue des gestionnaires vis-à-vis la qualité.

À la lumière de cette analyse, nous pouvons conclure que l'ensemble des attributs de qualité identifié par Kitchenham et Walker (1986) semble correspondre de façon assez précise avec le cadre de Garvin (1984). De ce fait, Kitchenham et Walker (1986) stipulent qu'il n'est pas surprenant que les chercheurs en qualité logicielle aient certaines difficultés à imposer une définition et une structure communes sur les attributs de qualité avec de telles caractéristiques tellement différentes.

5.5 Classification des attributs externes de Kitchenham (1987)

Bien que l'analyse de Kitchenham (1987) sur les attributs liés à la rectification ou l'amélioration de logiciel existant (présentée à la section 5.3) suggère une façon d'augmenter l'indépendance entre les attributs, il est encore difficile d'arriver à un rationnel pour inclure ou exclure un attribut en particulier. Une procédure développée par Kitchenham (1987), qui prétend être utile pour l'identification des attributs à inclure dans un modèle de qualité, est de grouper les attributs de qualité potentielle en terme de certaines caractéristiques.

Ces caractéristiques sont présentées ci-dessous :

- les attributs qui s'appliquent seulement à certains types spécialisés de système (i.e. : intégrité, généralité) ;

- les attributs qui sont généraux pour la plupart des types de système mais qui doivent être définis en respectant l'application (i.e. : efficacité et utilisabilité (en terme de facilité d'utilisation)) ;

- les attributs qui sont généraux pour la plupart des types de système et pouvant être définis d'une façon indépendante de l'application (i.e. : fiabilité, maintenabilité, réutilisabilité, extensibilité et utilisabilité (en terme d'acceptance de l'usager)) ;

- les attributs qui sont liés à la production logicielle (i.e. : testabilité, compréhensibilité et justesse) ;

Chacune des catégories est décrite en détails dans les sous-sections 5.5.1.1 à 5.5.1.4. Toutefois, l'évaluation subjective qui est présentée dans ces sous-sections doit être sujette à une évaluation empirique.

La classification proposée est malheureusement basée sur un ensemble restreint d'attributs de qualité. Bien que cet ensemble inclut les principaux attributs de qualité, de nouveaux attributs qui émergent, tels la contribution du système à l'organisation et l'ergonomie du logiciel, ne sont pas considérés. À notre avis, en plus de fournir un guide pour l'identification des attributs à inclure dans un modèle de qualité, cette classification permet d'adapter le modèle de qualité pour chacune des nouvelles applications en identifiant quels attributs se devront d'être modifiés et quels attributs pourront rester tels quels.

5.5.1.1 Qualités spécifiques à certaines applications

L'attribut *intégrité* est une caractéristique des systèmes qui ont des exigences strictes de sécurité. L'attribut *généralité* est seulement nécessaire pour les systèmes qui risquent d'être interfacés avec différents matériels et systèmes logiciels. Ces attributs de qualité correspondent aux caractéristiques d'un système en particulier. Ainsi, le type de critères utilisés pour définir ces attributs de qualité devrait correspondre aux caractéristiques incorporées dans le système au lieu de propriétés mesurables du système.

5.5.1.2 Qualités générales nécessitant une définition en rapport avec l'application

Les attributs *efficacité* et *utilisabilité* sont des qualités attendues de la plupart des systèmes, mais les caractéristiques d'un système qui fournit ces qualités sont dépendantes des systèmes. Le concept d'*efficacité* comme qualité logicielle est souvent chevauché avec celui de la performance. Lorsqu'elle est prise comme une indication du niveau de performance d'un système, cette mesure variera d'un système à l'autre.

L'attribut *utilisabilité* en terme des caractéristiques fournies pour faciliter l'utilisation d'un système dépend de l'utilisateur auquel le système est destiné. Cette vision de l'utilisabilité est concentrée sur les dispositions des caractéristiques spécifiques dans un système. De telles caractéristiques peuvent être vérifiées durant les tests du système et d'acceptation.

Des qualités de ce type nécessitent une redéfinition pour chaque système individuel (ou peut-être classe de systèmes). Ainsi, la recherche de critères généraux de qualité semble inappropriée. Les attributs *utilisabilité* et *efficacité* sont des exemples d'attributs de qualité qui devraient être approchés en utilisant la technique de spécification de qualité de Gilb (1986).

5.5.1.3 Qualités générales dont la définition est indépendante de l'application

La *fiabilité* est l'exemple le plus connu d'un attribut externe de qualité qui a une définition mesurable et indépendante de l'application : la probabilité qu'un système fonctionne sans incident pour une période de temps spécifiée sous des conditions spécifiées. Il est à noter que, pour le logiciel. il n'y a pas d'implication que la fiabilité soit une propriété statique d'un système. En fait, on s'attend plutôt à l'inverse puisqu'il est généralement admis que, lorsque des fautes sont découvertes et enlevées du logiciel, sa fiabilité inhérente augmente.

Il est important, lorsque possible, pour les autres attributs de qualité d'être également définis d'une manière indépendante de l'application.

L'attribut *maintenabilité* devrait être considéré comme un concept multidimensionnel puisqu'il est vu différemment par les producteurs et les utilisateurs de logiciels. Les utilisateurs de logiciels considèrent un système bien maintenu comme en étant un dont les rapports d'événements sont traités dans les plus courts délais tandis que les producteurs de logiciels sont généralement plus concernés à traiter les rapports

d'événements avec un minimum d'effort. De plus, un système bien maintenu devrait jouir d'une faible probabilité de générer des réponses incorrectes aux interrogations. Ainsi, l'attribut *maintenabilité* peut être défini en terme du *temps* pour diagnostiquer une faute pour n'importe quel défaut identifié, de l'*effort* pour diagnostiquer et réparer et la probabilité d'un diagnostic et d'une réparation réussis.

En considérant les attributs *maintenabilité* et *fiabilité* comme étant des qualités qui peuvent être évaluées en fonction des rapports d'événements des utilisateurs, il vient à l'idée que l'attribut *utilisabilité* (en terme du niveau de qualité perçu par l'usager) peut être mesuré d'une façon indépendante des applications. Ainsi, une mesure de la satisfaction (insatisfaction) des usagers peut être obtenue en considérant la proportion ou le taux de rapports d'événements qui ne correspondent pas à des défauts (logiciel, matériel ou documentation). Cette vue de l'*utilisabilité* permet que son évaluation soit considérée comme un cas spécial de *fiabilité* et évaluée comme faisant partie des activités de maintenance.

L'attribut *extensibilité* peut être défini simplement en terme de l'effort et du temps nécessaires pour faire certaines classes d'extension à un système existant. Les classes d'extension incluent l'incorporation de nouvelles caractéristiques, l'extension des caractéristiques existantes ou la correction des erreurs du système. Il peut également être approprié de considérer l'attribut *extensibilité* comme une mesure relative comparant la productivité d'étendre un système existant versus la productivité d'un système autonome.

L'attribut *réutilisabilité* peut être mesuré en terme de l'effort dépensé à produire une composante logicielle particulière ajustée par un estimé du temps moyen requis pour utiliser une composante déjà existante ainsi que le nombre de fois que la composante peut être réutilisée dans d'autres systèmes.

Toutes ces qualités générales peuvent être mesurées de façon directe mais seulement durant la vie active du produit. Ainsi, même si ces qualités peuvent être spécifiées quantitativement, elles ne peuvent pas être démontrées ou vérifiées directement tant que

le produit n'est pas dans les mains du client. Bien que ces qualités ne puissent être directement évaluées qu'après la diffusion du logiciel, cela n'implique pas que l'atteinte de ces qualités soit seulement un acte de foi. Certaines techniques visant l'atteinte de la qualité sont disponibles pour les groupes de production logicielle. Ceci implique qu'au lieu d'utiliser des critères de qualité pour définir ces qualités, il serait préférable d'identifier les caractéristiques du produit et les techniques de développement qui aident à atteindre un certain niveau de qualité. Cette suggestion est consistante avec la vision de Gilb (1986) qui veut que les qualités logicielles soient liées à des techniques de génie logiciel.

5.5.1.4 Qualités liées au processus de production logicielle

L'attribut *testabilité* est une qualité liée aux caractéristiques du design et au processus de développement. Cet attribut est nécessaire pour s'assurer qu'un produit peut être testé correctement, c'est-à-dire que toutes ses qualités et capacités fonctionnelles peuvent être examinées d'une façon critique et comparées avec les exigences.

L'attribut *compréhensibilité* est lié à la clarté et l'intelligibilité des exigences variées, du design, du code et des documents de planification ou de listage produits durant le développement logiciel.

L'attribut *justesse* est lié à la nature inhérente du logiciel - l'atteinte de cet attribut est le but de la vérification : assurer la conformité aux spécifications.

Ces qualités semblent être les seules que le groupe de production ait sous son contrôle direct. Elles semblent être adéquates pour une définition basée sur un attribut, schéma de liste de contrôle où les métriques liées à la liste de contrôle peuvent être obtenues à différentes phases dans le processus de développement pour suivre le progrès du produit.

En plus des critères qui sont liés aux qualités individuelles, une caractéristique qui affecte tous les attributs est la *stabilité* du produit. Si les exigences et les décisions de design sont modifiées durant les étapes subséquentes du développement, il est alors très difficile de préserver la *justesse*, la *testabilité* et la *compréhensibilité* d'un produit.

Toutefois, il demeure important d'assurer que l'atteinte d'un logiciel testable, compréhensible et exact résultera en l'atteinte des qualités telles la *fiabilité*, la *maintenabilité*, l'*utilisabilité* et l'*extensibilité* qui peuvent seulement être expérimentées lorsque le produit est dans les mains du client.

Une évaluation subjective des relations entre les qualités liées au développement et les qualités liées au produit final suggère que l'atteinte de la *fiabilité* est supportée par la *justesse* (par la vérification et la validation) ; l'attribut *maintenabilité* est lié à la fois à la *testabilité* (par les caractéristiques de dévermillage telles le traçage et les bancs d'essais) et à la *compréhensibilité* (par la documentation des exigences, du design et du code) ; et finalement les attributs *extensibilité* et *réutilisabilité* sont supportés par la *compréhensibilité* à la fois de la documentation et de l'implantation en termes du design et de la structure des modules. Ainsi, les qualités liées au développement agissent presque comme les critères sous-jacents aux qualités liées au produit final et les métriques de qualité peuvent être vues comme des indicateurs des valeurs des attributs de qualité finale.

5.6 Classification des attributs externes développée

Ayant étudié une dizaine de modèles de qualité logicielle, nous nous sommes intéressés à développer une classification des attributs externes de ces modèles. Cette classification sera d'une aide précieuse quand viendra le moment de choisir les attributs de notre modèle de qualité.

Les regroupements dans la classification ne se font pas au niveau des attributs mais au niveau des concepts décrivant les attributs. Ainsi, un même attribut peut se retrouver dans plusieurs groupes différents. Par exemple, l'attribut *maintenabilité* se retrouve dans deux groupes selon que le concept décrivant la *maintenabilité* réfère à l'évolution du logiciel ou simplement à la correction du logiciel.

La classification s'étendant sur plusieurs dizaines de pages, elle est présentée à l'annexe III. Au total, la classification comporte plus de deux cents attributs d'une dizaine de modèles de qualité logicielle.

CHAPITRE VI
LES ATTRIBUTS INTERNES DE QUALITÉ : LES MÉTRIQUES LOGICIELLES

Au cours des dernières années, plus d'un millier de métriques logicielles ont vu le jour. Or, comme nous le verrons à la section 6.1, certaines de ces métriques sont plus ou moins pertinentes. La première utilisation qui nous vient à l'esprit lorsque nous pensons aux métriques logicielles est l'évaluation de la qualité finale des produits. Or, les métriques peuvent également être utilisées dans un tout autre contexte (section 6.2). Le choix des métriques est une activité primordiale puisque les métriques interagissent directement avec les éléments dont nous voulons évaluer ou prédire la qualité. Certains points à considérer lors de ce choix sont présentés à la section 6.3.

Il pourrait sembler pertinent de présenter les métriques existantes les unes à la suite des autres, mais de l'avis de plusieurs auteurs, dont Boloix et Robillard (1994B), une telle énumération serait inadéquate puisqu'il est essentiel de fournir le contexte dans lequel les métriques sont utilisées pour comprendre leur implication. Ainsi, nous ne dissocierons pas les attributs externes des modèles de qualité des métriques de qualité. Nous vous référons à l'annexe B pour une présentation des métriques utilisées en conjoncture avec les modèles de qualité. Toutefois, la section 6.4 présente un aperçu de certaines métriques majeures avec une brève description de leur portée.

6.1 Situation actuelle des métriques logicielles

Au dire de Arthur et al. (1993), plusieurs des métriques existantes sont non-intuitives, non-instructives et manquent une base fondamentale pour comprendre leur implication comparée à d'autres mesures.

Une révélation encore plus frappante est qu'il apparaît que plusieurs métriques sont invalides ou dénudées de sens selon la théorie de la mesure (Fenton, 1991). Un de ces exemples est la métrique fort connue de Boehm (1981), soit le modèle de coût COCOMO, qui enfreint certaines règles élémentaires de la théorie de la mesure.

Également, plusieurs métriques prétendent mesurer certaines caractéristiques du logiciel. Malheureusement, de telles affirmations sont souvent gratuites et manquent l'expérimentation fondamentale pour comprendre leur implication. Pour nous convaincre de la fébrilité du domaine, ISO-9126 (1991) n'a pas inclus de métriques dans son modèle de qualité puisqu'il croit que l'état de l'art n'est pas encore assez avancé.

À la lumière de ces dernières constatations, il faut être extrêmement vigilant lors du choix des métriques.

6.2 Les métriques logicielles comme support de développement

Bien que les métriques soient très utiles pour évaluer la qualité finale des produits logiciels, elles peuvent jouer également un autre rôle tout aussi important. Ainsi, elles peuvent servir à contrôler les processus de développement et de maintenance logiciels.

Dans cet ordre d'idées, les métriques fournissent l'information de retour nécessaire pour juger l'efficacité de plusieurs activités reliées aux processus et soutiennent la prise de décision pour l'amélioration des processus. Il est proposé de les utiliser tout au long du développement logiciel. Les premières métriques sont basées sur les caractéristiques du processus puisqu'il n'y a que peu ou pas de produits disponibles. Lorsque le développement évolue et que les produits deviennent disponibles, de nouvelles métriques devraient être utilisées pour refléter la qualité au niveau des

produits. La Figure 6.1 illustre graphiquement ces dernières constatations. Une étude préliminaire, réalisée par Arthur et al. (1991) dans le domaine des métriques, suggère que des métriques de processus, de documentation et de code sont nécessaires.

Il est intéressant de remarquer que pour évaluer la qualité des produits intermédiaires à chacune des phases du cycle de développement, les développeurs doivent utiliser différentes métriques pour les mêmes caractéristiques puisque les mêmes métriques ne sont pas applicables à toutes les phases du cycle de vie.

L'utilisation des métriques comme support de développement relève de l'aspect prédictif de la qualité logicielle. Ainsi, les métriques servent à prédire la qualité logicielle.

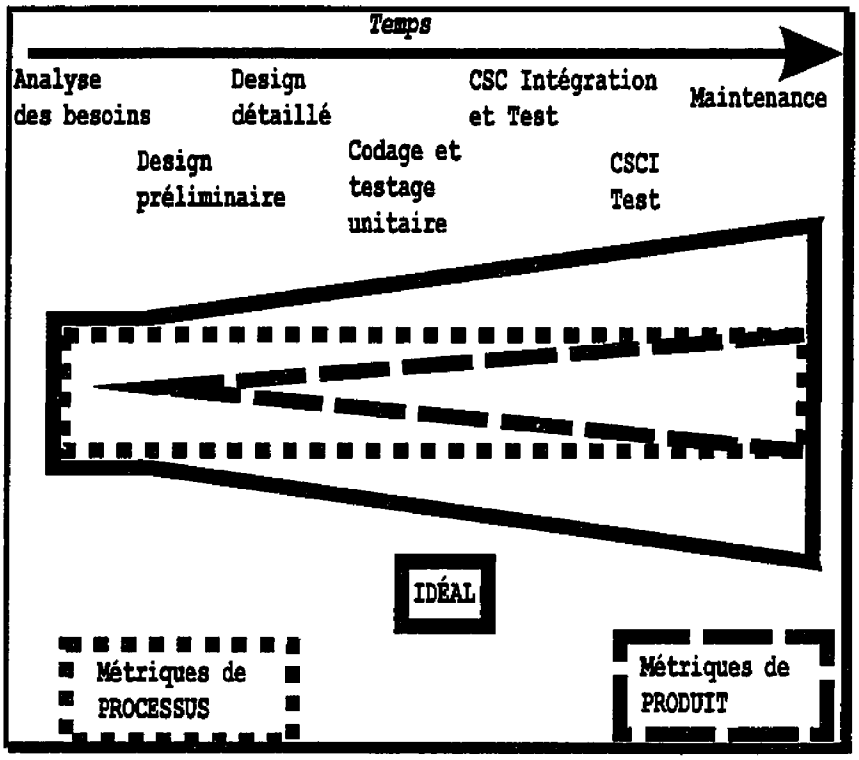


Figure 6.1 : exploitation des métriques de processus et de produit

6.3 Identification des métriques de qualité logicielle

L'identification des métriques de qualité logicielle consiste à choisir des métriques (ou en élaborer) qui permettront de mesurer le degré d'atteinte des attributs externes du modèle de qualité et cela à différentes étapes du cycle de vie. Or, comme nous l'avons déjà mentionné, cette activité est excessivement délicate puisqu'il n'y a pas de corrélation absolue entre les attributs externes et les attributs internes (métriques). Un exemple typique illustrant cette difficulté est soulevé par Kitchenham (1987) : il n'y a pas de relation fonctionnelle évidente entre le nombre de fautes trouvées durant les tests

et la fiabilité du produit final. Il existe cependant, des méthodes reconnues pour évaluer la pertinence des métriques qui sont l'objet de l'annexe IV.

6.4 Présentation des métriques majeures

Cette section a pour but de présenter certaines des métriques majeures que nous retrouvons dans la littérature. Le Tableau 6.1, issu de IEEE 1061-1992 (1992), présente ces métriques avec une description de leur portée.

Tableau 6.1: analyse des métriques majeures selon IEEE std. 1061-1992

Métriques	Description	Avantages	Limitations	Considérations d'application
Halstead - software science ("effort and difficulty") (Halstead, 1977)	Mesurer les propriétés et la structure d'un programme pour prédire les éléments suivants pour un programme : • Longueur, volume, difficulté et niveau ; • nombre d'erreurs ; • effort et temps requis pour le développement	• assez facile à calculer ; • peut être automatisé ; • certains outils existent	• ne peut pas être calculé tant que le logiciel n'est pas conçu ; • il y a une controverse sur sa validité ; • ne représente pas tous les aspects de la complexité	Une corrélation peut être démontrée entre un haut degré de difficulté et un nombre élevé d'erreurs et un effort de maintenance soutenu. Une corrélation similaire peut être démontrée avec des programmes très gros
Boehm - constructive cost model (COCOMO) (Boehm 1981)	Estimer les principaux facteurs de coût et les lignes de codes pour prédire l'effort d'un projet, le coût et l'échéancier			L'utilisation du modèle COCOMO peut fournir des estimés plus précis du coût et de l'échéancier lorsqu'il est utilisé sur une base logique avec de la formation

Albretch - function points (Albretch, 1983)	Mesurer les fonctions qu'un logiciel doit exécuter à partir des exigences pour estimer l'effort requis pour développer le *logiciel*			Les points de fonction d'Albretch peuvent être mesurés plus rapidement dans le cycle de vie que certaines autres métriques. Le nombre de points de fonction peut être corrélé avec les *lignes de code et l'effort* requis pour le développement
McCabe - cyclomatic complexity (McCabe, 1976)	Mesurer la structure du flot de contrôle pour prédire les éléments suivants d'un programme : •difficulté de compréhension ; •probabilité des défauts du programme	•facile à calculer ; •il existe des outils ; •peut s'appliquer à la fois au code et au design préliminaire	•doit avoir une idée de la structure du design avant qu'elle ne soit appliquée ; •il y a une controverse sur sa validité ; •ne représente pas tous les aspects de la complexité	L'utilisation de la métrique de complexité cyclomatique de McCabe peut aider à s'assurer que les développeurs reconnaissent les programmes plus difficiles à comprendre et ceux pouvant contenir des défauts. Les résultats peuvent être corrélés avec les lignes de codes et aussi avec le nombre de défauts
Source d'erreur logicielle	Le comptage des erreurs par type	Lie le problème à la source de l'erreur pour que le problème puisse être corrigé	Difficile à automatiser et difficile d'identifier la source de l'erreur	
Nombre de changements qui affectent un module	Le comptage des modifications d'un module	•facile à calculer ; •peut être automatisé ; •il existe des outils	Ne peut pas être utilisée avant que le logiciel n'existe	
Défauts	Le comptage des erreurs, le temps entre des erreurs, le *taux d'erreur, le* comptage des corrections d'erreurs et le temps requis pour corriger les erreurs	Peut évaluer la fiabilité et l'approche du *design*	Les données peuvent être dures à cueillir	

Traçage	L'identification des chemins indépendants correspondant à la métrique de McCabe pour les tests des chemins	Excellent pour évaluer si la documentation et le logiciel sont complètement intégrés	• sujet à des interprétations variées ; • difficile à définir quantitativement ; • difficile à automatiser	
Nombre de chemins d'un programme (Np)	Nombre de séquences distinctes d'exécution d'un noeud d'entrée à un noeud de sortie	Représentatif des séquences d'exécution dans un programme	Peut être difficile à calculer puisque le nombre de chemins peut-être très grand. Le temps de calcul peut-être excessif	

CHAPITRE VII
AGRÉGATION DES DONNÉES

Ce chapitre a pour objet la description des méthodes d'agrégation de données. Nous présenterons en premier lieu ce qu'est une méthode d'agrégation et dans quel contexte nous l'utilisons (section 7.1). Nous verrons ensuite quelles sont les principales qualités recherchées de telles méthodes (section 7.2). Par la suite, nous présenterons les méthodes qui sont utilisées en conjoncture avec les modèles de qualité (section 7.3). Bien qu'il existe de nombreuses méthodes d'agrégation, nous en présentons seulement deux : le processus d'analyse hiérarchique (section 7.4) et une approche par règles (section 7.5). Nous concluons le chapitre avec une comparaison entre les deux méthodes qui sont proposées (section 7.6).

7.1 En quoi consiste une méthode d'agrégation de données ?

Une méthode d'agrégation de données est une procédure permettant de regrouper, selon un certain algorithme, un ensemble de données en un autre ensemble étant plus concis et captant l'essence de l'ensemble initial. Ainsi, une méthode d'agrégation a comme principale utilité de regrouper un certain nombre d'éléments pour en faciliter la gestion intellectuelle.

En appliquant cette définition générale au domaine de la modélisation de la qualité logicielle, nous déduisons qu'une méthode d'agrégation de données logicielles est une procédure agrégeant les données des niveaux inférieurs vers les niveaux supérieurs.

7.2 Qualités recherchées d'une méthode d'agrégation

Les principales qualités recherchées d'une méthode d'agrégation de données pour notre domaine d'étude sont les suivantes :

- Elle doit propager le plus fidèlement possible l'essence des données lors de leur agrégation. Une méthode qui agrège les données en perdant une partie importante de la sémantique de ces données est à éviter puisqu'elle donnera une fausse représentation de la qualité d'un produit;

- Être compatible avec le type d'architecture du modèle de qualité qui a été choisi. Il faut s'assurer que la méthode fonctionne avec l'architecture, particulièrement dans le cas où cette architecture permet le chevauchement des attributs d'un même niveau;

- Être capable de gérer à la fois des données quantitatives et qualitatives. Cette caractéristique est importante puisqu'il n'est pas encore possible au dire de Finie et al. (1993) de caractériser un logiciel seulement à l'aide de données quantitatives;

- Être capable d'affecter des priorités entre les attributs de qualité. Comme nous l'avons déjà mentionné à la section 3.3, il est évident que pour une certaine classe de logiciels, les attributs n'ont pas tous la même importance;

- Respecter la théorie de la mesure lors d'opérations mathématiques sur les données. Des opérations mathématiques invalides peuvent mener à une fausse représentation de la qualité d'un produit;

- Elle doit s'adapter facilement à une modification de l'architecture (retrait ou ajout d'un sommet et/ou d'un ou plusieurs arcs).

7.3 Méthodes d'agrégation utilisées dans la littérature

L'étude des modèles de qualité selon la méthode d'agrégation de données s'est avérée plus ou moins profitable. Mis à part le modèle de qualité de Boloix et Robillard (1994A) et le modèle de productivité de Finnie et al. (1993) qui utilisent le processus d'analyse hiérarchique, les autres modèles semblent très vulnérables à ce niveau. Ainsi, la majorité des modèles ne font qu'effleurer l'activité d'agrégation. Un exemple typique est le cas du standard ISO-9126 (1991) qui mentionne simplement que l'évaluation de la qualité est déterminée en assignant une pondération à chacune des caractéristiques individuelles de leur modèle. Malheureusement, il semble que de tels modèles sont à priori difficilement utilisables puisque nous n'avons aucune indication claire de la façon de manipuler les valeurs des attributs qui les composent.

7.4 Processus d'analyse hiérarchique ("AHP - Analytic Hierarchical Process")

La technique AHP a été développée dans les années 1970 par Saaty (1990) comme étant une technique d'aide à la décision basée sur des critères multiples. Il semble que cette technique soit de plus en plus populaire puisqu'elle a été utilisée dans de nombreuses situations. De plus, Schoemaker et Waid (1982) ont démontré que cette technique était la meilleure des cinq qui furent évaluées sur la détermination des poids dans les modèles additifs utilitaires.

D'une façon sommaire, la technique est la suivante :

1. Définir le problème hiérarchiquement où les niveaux les plus hauts reflètent les objectifs de gestion et les niveaux les plus bas reflètent les attributs qui les influencent. Ce qui correspond essentiellement à l'élaboration du modèle de qualité, soit les trois premières étapes de la procédure qui a été développée à la section 3.5.2.

2. Créer des matrices de comparaison entre chaque groupe de deux éléments d'un même niveau de la hiérarchie pour leur contribution ou leur influence sur un attribut de plus haut niveau qui leur est relié. Dans cette veine, une échelle de comparaison supposée optimale, proposée par Saaty (1990), est donnée dans le Tableau 7.1.

Tableau 7.1 : échelle de comparaison entre les attributs

1	Importance égale
3	Importance modérée d'un envers l'autre
5	Importance essentielle ou forte
7	Importance très forte et démontrée
9	Importance absolue
2,4,6,8	Valeurs intermédiaires entre les différentes échelles de valeur

Ces comparaisons entre paires de n éléments sont résumées dans la matrice A qui est donnée par :

$$A = \begin{bmatrix} w_1/w_1 & w_1/w_2 & \ldots\ldots & w_1/w_n \\ w_2/w_1 & w_2/w_2 & \ldots\ldots & w_2/w_n \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ w_n/w_1 & w_n/w_2 & \ldots\ldots & w_n/w_n \end{bmatrix} \quad (1)$$

Les informations exhibées dans la matrice sont interprétées de la façon suivante : chaque élément, a_{ij}, de la matrice A montre la contribution relative pour le sujet de comparaison de la $i^{ème}$ activité comparée à la $j^{ème}$ activité, i.e.

$$a_{ij} = w_i / w_j \qquad 1 \le i \le n, 1 \le j \le n \qquad (2)$$

Noter que

$$Aw = nw \text{ où } w^t = [w_1, w_2, \ldots, w_n] \quad (3)$$

(w^t étant la matrice transposée de w)

Le vecteur de priorité, w, est obtenu de l'équation (3) et correspond au vecteur propre normalisé de la matrice A correspondant à la plus grande valeur propre $L_{max}=n$.

3. Lorsqu'il y a des inconsistances, i.e., $a_{ik} \neq a_{ij} \bullet a_{jk}$, cette valeur propre la plus grande est supérieure à la dimension de la matrice, n. Un index de consistance est défini par $CI=[L_{max}-n]/[n-1]$. Saaty (1990) définit une mesure appelée le ratio de consistance comme étant le ratio de l'index de consistance sur l'index de consistance moyen d'une matrice réciproque du même ordre générée aléatoirement. L'utilisation de ce ratio identifie les comparaisons où une révision du jugement est nécessaire ; ce qui est fait lorsque le ratio de consistance est plus grand que 0,1.

4. Lorsqu'il y a plus d'un niveau impliqué, la composition hiérarchique est utilisée pour pondérer les vecteurs propres par les poids des critères et la somme est prise sur tous les vecteurs propres pondérés des niveaux inférieurs, et ainsi de suite. Ces opérations résulteront en un vecteur de priorité global pour le niveau le plus bas de la hiérarchie.

L'explication du processus d'analyse hiérarchique étant très théorique, un exemple d'utilisation permettra de mieux comprendre les principes en jeu ; cet exemple fait l'objet de la prochaine sous-section.

7.4.1 Exemple d'utilisation du processus d'analyse hiérarchique

Pour démontrer l'utilisation du processus d'analyse hiérarchique, nous utiliserons une architecture typique de modèle de qualité qui est présentée à la Figure 7.1. Cette architecture est une adaptation de l'architecture facteur-critère-métrique.

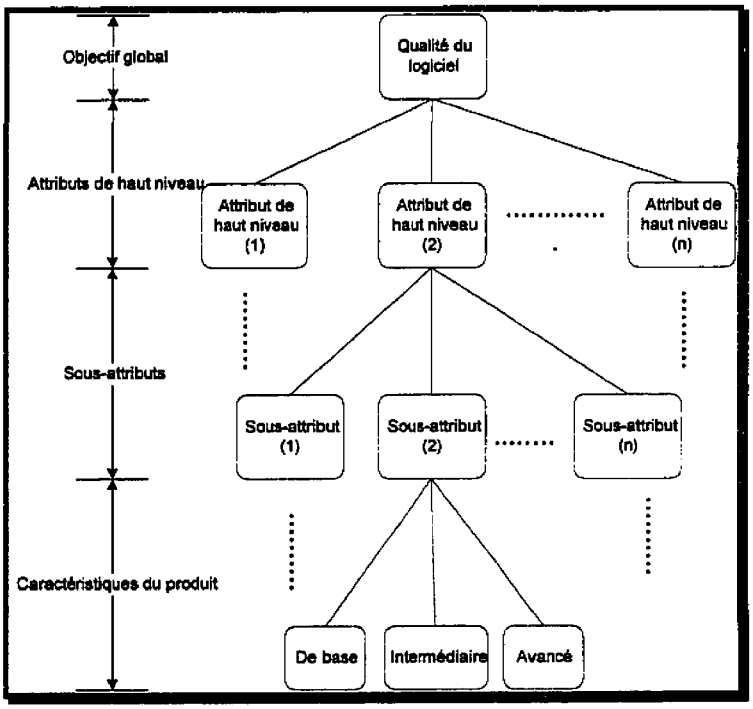


Figure 7.1 : architecture typique d'un modèle de qualité

La première étape est d'évaluer les priorités entre les attributs de haut niveau. Le Tableau 7.2 présente cette évaluation. Il est à noter que les chiffres sont fictifs et n'ont aucune signification.

Tableau 7.2 : importance des attributs de haut niveau

	1	1/3	1/5
	3	1	1/3
	5	3	1

Maintenant, pour chaque attribut de haut niveau, il faut faire une évaluation des priorités entre les sous-attributs correspondants. Le Tableau 7.3 présente cette évaluation pour l'*attribut de haut niveau (2)*.

Tableau 7.3 : importance des sous-attributs de l'attribut de haut niveau (2)

	1	5	7
	1/5	1	3
	1/7	1/3	1

Le dernier niveau de la hiérarchie correspond à l'importance relative des niveaux de classement des catégories (de base, intermédiaire, avancé). Le Tableau 7.4 présente cette évaluation pour le *sous-attribut (2)*.

Tableau 7.4 : importance des niveaux de classement pour le *sous-attribut (2)*

	1	1/3	1/5
	3	1	1/3
	5	3	1

Chaque matrice génère un vecteur propre (calculé en normalisant les valeurs par colonne et en prenant la moyenne par ligne) qui classe l'importance relative des éléments à chaque niveau de la structure. En utilisant ce vecteur propre comme un facteur de pondération, chaque attribut du système évalué est pondéré. Les résultats sont composés niveau par niveau pour avoir une évaluation globale du système logiciel.

Le Tableau 7.5 présente un exemple de calcul de vecteur propre pour les attributs de haut niveau.

Tableau 7.5 : vecteur propre pour la matrice des attributs de haut niveau

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	(1)/[9]	(1/3)/[13/3]	(1/5)/[23/15]	0.1062
[illegible]	(3)/[9]	(1)[13/3]	(1/3)/[23/15]	0.2605
[illegible]	(5)/[9]	(3)[13/3]	(1)/[23/15]	0.6333
[illegible]	9	13/3	23/15	

Ainsi, après avoir pondéré chacun des attributs de chaque niveau, on obtient le portrait global du système qui pourrait ressembler à ce qui est présenté dans le Tableau 7.6. Un exemple concret de l'utilisation de cette méthode est fourni dans (Boloix et Robillard, 1994B).

Tableau 7.6 : portrait global de l'évaluation du système fictif selon la technique AHP

[illegible]	[illegible]
[illegible]	0,24
[illegible]	0,34
[illegible]	0,48
[illegible]	0,37

7.4.2 Évaluation du processus d'analyse hiérarchique selon les qualités désirables

Cette méthode semble posséder la majorité des qualités désirables que nous avons élaborées à la section 7.2. Ainsi, la méthode propage d'une façon cohérente les données des niveaux inférieurs vers les niveaux supérieurs tout en respectant la théorie de la mesure. Elle est compatible avec tous les types d'architecture. Elle peut gérer

simultanément des données quantitatives et qualitatives et également affecter des priorités entre les attributs.

Cependant, il semble que la méthode soit plus ou moins flexible pour ce qui est de la modification de l'architecture. Par exemple, l'ajout d'un sommet et de l'arc ou des arcs qui lui sont incidents oblige à reconsidérer tous les vecteurs propres qui ont été préalablement calculés.

7.5 Approche par règles

Cette approche est présentée seulement pour des fins de comparaison avec le processus d'analyse hiérarchique. Ainsi, elle n'est pas utilisée en pratique puisqu'elle donne des résultats trop vagues.

Pour démontrer l'approche par règles, nous allons utiliser encore une fois l'architecture typique d'un modèle de qualité qui fut l'objet de la Figure 7.1. Dans cette approche, l'assignation des catégories à un certain niveau de la hiérarchie suit une approche par règles. Par exemple, l'assignation d'une catégorie à l'*attribut de haut niveau (2)* aurait l'apparence suivante :

IF *sous-attribut (1)* = « de base »
ET *sous-attribut (2)* = « intermédiaire »
ET *sous-attribut (3)* = « intermédiaire »
ALORS *attribut de haut niveau(2)* = « bas »

Pour chacun des niveaux de la hiérarchie, il faut construire l'ensemble des règles qui couvrent toutes les possibilités. Évidemment, l'assignation des catégories est purement empirique et doit être réalisée par jugement d'experts. Ainsi, l'approche par règles peut être qualifiée comme étant une assignation symbolique au lieu d'être quantitative. Le

Tableau 7.7 présente un exemple d'assignation des catégories pour l'*attribut de haut niveau (2).*

Tableau 7.7 : assignation des catégories pour *l'attribut de haut niveau (2)*

B-B-B	I-I-I	A-A-A
B-I-B	B-A-I	A-I-A
B-B-I	I-B-A	I-A-A
I-B-B	A-B-I	A-A-I
I-B-I	I-A-B	A-I-I
B-I-I	B-I-A	I-A-I
I-I-B	A-I-B	I-I-A
B-B-A	A-A-B	
B-A-B	B-A-A	
A-B-B	A-B-A	

À titre d'exemple, le Tableau 7.8 présente l'évaluation fictive du système présenté à la section 7.4, mais cette fois-ci en utilisant l'approche par règles.

Tableau 7.8 : portrait global du système fictif avec l'approche par règles

Attribut de haut niveau	Bas
Attribut de haut niveau (2)	Bas
Attribut de haut niveau (3)	Haut
Évaluation globale	Intermédiaire

7.6 Comparaison entre les méthodes AHP et par règles

Évidemment, la méthode AHP est beaucoup plus puissante que celle par règles puisqu'elle fournit plus d'informations sur les résultats de l'évaluation. L'approche par règles ne considère pas l'importance relative des attributs, ce qui produit des assignations plutôt dénudées de sens. Le mérite de l'approche hiérarchique est qu'elle utilise une échelle *ratio*, alors que l'approche par règles utilise une échelle *ordinale*.

Troisième partie : le modèle de qualité logicielle développé

CHAPITRE VIII
LE MODÈLE DE QUALITÉ LOGICIELLE DÉVELOPPÉ

Ce chapitre est le cœur du présent document, son objectif est de présenter le modèle de qualité logicielle que nous avons développé en se référant aux éléments dont nous avons préalablement discutés dans les première et deuxième parties du document.

Les deux premières sections du chapitre décrivent les orientations majeures du modèle, soit sa portée (section 8.1) ainsi que le ou les points de vue qui sont considérés (section 8.2).

Les orientations ayant été définies, il est maintenant possible de réaliser rigoureusement chacune des étapes de l'élaboration d'un modèle de qualité, étapes qui ont été vues au chapitre 3. Le choix de l'architecture logicielle sur laquelle notre modèle est basée est décrit à la section 8.3. Ayant choisi l'architecture, les deux prochaines étapes consistent à la peupler, c'est-à-dire choisir les attributs externes et internes du modèle de qualité (section 8.4 et 8.5). La dernière étape est de choisir ou d'élaborer une procédure qui permet d'évaluer la qualité du logiciel en se basant sur le modèle, ce qui est l'objet de la dernière étape, soit le choix d'une méthode d'agrégation des données (section 8.6).

Les phases de l'élaboration ayant été toutes réalisées, il serait intéressant d'utiliser notre modèle dans une situation réelle, ce qui est l'objectif de la dernière section (section 8.7).

8.1 Portée du modèle

Notre objectif est de développer un modèle de qualité logicielle général, modèle qui est applicable à toutes les classes d'applications logicielles. Ce choix est guidé par des considérations à la fois économiques et d'ordre organisationnel. La première contrainte est qu'il s'avérerait excessivement long de développer un modèle de qualité pour chacune des classes d'applications. La deuxième contrainte est que notre partenaire industriel est intéressé par des applications logicielles qui touchent non pas une classe particulière mais plutôt un ensemble de classes d'applications et que le processus pour supporter plusieurs modèles serait trop lourd.

Évidemment, le choix de développer un modèle de qualité général a des répercussions au niveau de l'élaboration. Plus spécifiquement, deux phases s'en trouvent affectées. Ces phases sont celles du choix des attributs externes et des attributs internes qui doivent prendre en considération le caractère général du modèle. Ainsi, les attributs spécifiques à certaines classes d'applications devront être laissés de côté.

8.2 La nécessité de reconnaître les différentes vues de la qualité

Comme nous en avons discuté à la section 1.3, il y a plusieurs vues de la qualité. Garvin (1984) en définit cinq, alors que ISO-9126 (1991) en définit trois. Nous sommes du

même avis que ces auteurs et reconnaissons que la vision de la qualité dépend du point de vue qui est considéré.

Pour tenir compte de cette réalité, nous avons développé une version du modèle pour chacune des trois vues de la qualité reconnues par ISO-9126. Il y a donc une version pour les utilisateurs, une pour les réalisateurs et une dernière pour le maître d'ouvrage. Évidemment, il y a des chevauchements entre ces versions. L'union des trois versions du modèle forme ce que nous appelons le méta-modèle.

Une telle approche pour concilier les trois vues de la qualité est, à notre avis, une première. Plusieurs auteurs de modèle reconnaissent l'importance de ces vues, mais aucun d'eux ne porte d'action pour en tenir compte.

8.3 L'architecture logicielle choisie

Comme nous en avons discuté au chapitre 4, l'architecture choisie devra accommoder toute la connaissance relative à la qualité logicielle et cela, d'une façon constructive, raffinable et intellectuellement gérable. Ainsi, l'architecture doit être compréhensible à plusieurs niveaux, décomposable et adaptable.

Ces dernières exigences peuvent être rephrasées en des termes plus explicites :

- Les attributs peuplant le modèle doivent être regroupés dans une architecture qui interdit les chevauchements possibles. Cette exigence vise à éliminer les architectures qui sont difficilement gérables intellectuellement. Une architecture permettant le chevauchement n'est pas souhaitable puisqu'elle

dégénère très souvent en un fouillis incompréhensible. Pour vous en convaincre, jetez un coup d'œil sur les modèles de Arthur et al., de Dromey et de McCall et al. qui sont présentés à l'annexe II. Évidemment, une telle exigence a des répercussions qu'il ne faut pas négliger. Ainsi, que fait-on quand un critère de qualité semble être lié à plus d'un facteur de qualité ? Un compromis acceptable est de considérer uniquement la relation dominante ;

- L'architecture doit supporter un nombre d'attributs variables par niveau. Ce choix est purement un raisonnement par l'absurde qui stipule qu'il n'y a pas de raisons pratiques ou théoriques permettant de justifier que chaque attribut du niveau i doit dépendre du même nombre d'attributs du niveau $i+1$;

- Puisque la modélisation de la qualité logicielle est un domaine en constante évolution, il doit être facile de modifier le modèle pour tenir compte de nouvelles réalités. Pour cela, l'architecture doit supporter l'ajout et la suppression d'attributs ;

- Puisque certains attributs peuvent appartenir à des domaines différents et incompatibles de la qualité du logiciel, l'architecture ne doit pas forcer tous les attributs à être agrégés vers les mêmes attributs. Ainsi, il peut y avoir plus d'un attribut au niveau le plus haut de l'architecture.

En transposant ces dernières exigences dans la vocabulaire de la théorie des graphes, nous obtenons les exigences suivantes :

- Les sommets doivent être indépendants ;
- L'architecture ne fait pas partie de la famille des architectures N-aires ;
- L'architecture comporte plus d'une arborescence.

Une seule architecture répond à toutes ces exigences, selon l'analyse qui a été réalisée au chapitre 4; il s'agit de la famille des architectures *N arborescences*. Un exemple typique d'architecture faisant partie de cette famille est représenté à la Figure 8.1.

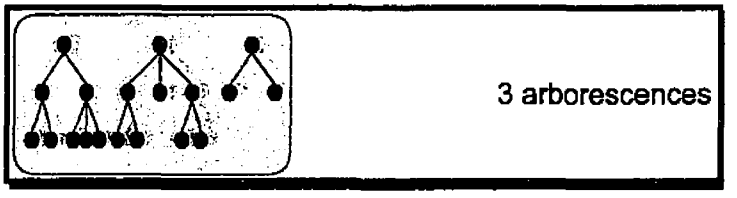


Figure 8.1: exemple d'architecture N arborescences

Cette architecture sera celle préconisée pour les trois versions du modèle de qualité ainsi que pour le méta-modèle.

8.4 Le choix des attributs externes

Le choix des attributs externes de qualité (facteurs et critères de qualité) est une activité excessivement critique, puisqu'elle cristallise notre vision de la qualité logicielle.

Pour être en mesure de choisir un ensemble d'attributs adéquats, nous nous appuyons principalement sur trois points :

- l'ensemble devra être complet;
- l'ensemble devra être non-entrelacé;
- l'ensemble devra être compatible.

La Figure 8.2 présente les attributs qui ont été retenus. Cette figure correspond au méta-modèle qui regroupe les trois vues de la qualité. Les sous-sections (section 8.4.1 à 8.4.3) décrivent en détail chacun des attributs du modèle en relation avec la vue qui est considérée.

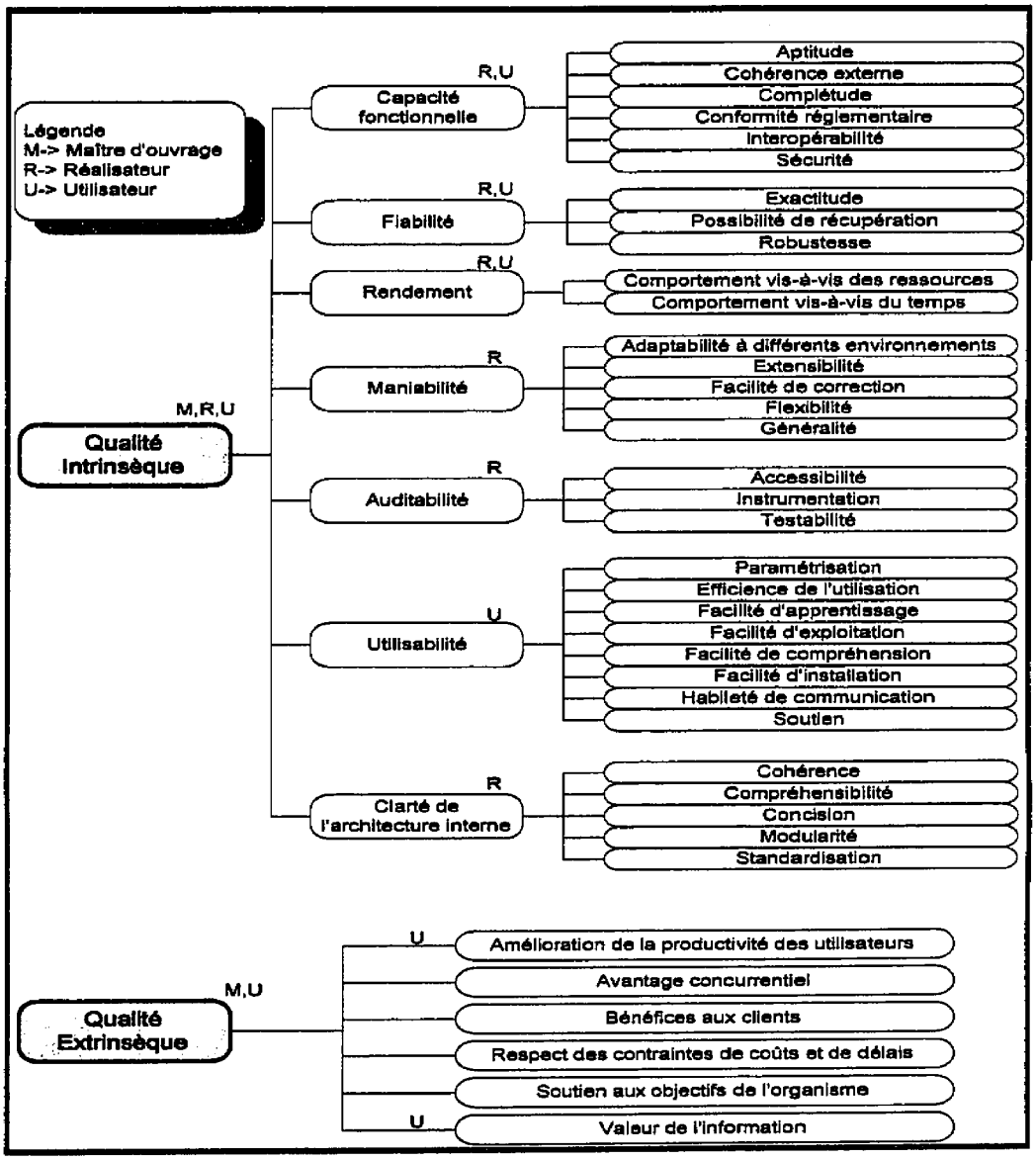


Figure 8.2: méta-modèle de qualité

Comme vous pouvez le remarquer sur la Figure 8.2, les attributs sont regroupés dans deux domaines distincts. Les attributs liés au domaine de la *qualité intrinsèque* sont ceux qui dépendent exclusivement du logiciel en faisant abstraction de l'environnement dans lequel ce dernier est opéré. À l'inverse, les attributs reliés au domaine de la *qualité extrinsèque* prennent explicitement en considération l'environnement dans lequel le logiciel est opéré.

L'ensemble d'attributs retenu répond aux trois points que nous avons énoncés précédemment. Ainsi, l'ensemble est complet puisqu'il intègre la vision de la qualité selon les trois perspectives possibles. Les attributs liés à une vue particulière de la qualité ne sont pas entrelacés. Finalement, les attributs d'une des vues de la qualité sont parfaitement compatibles avec les attributs des deux autres vues.

8.4.1 Vue des utilisateurs

Les utilisateurs sont principalement intéressés à utiliser le logiciel, ses performances et les effets d'utiliser le logiciel. Ils évaluent le logiciel sans en connaître les aspects internes.

De ce fait, les utilisateurs sont concernés par quatre facteurs de qualité liée au domaine de la qualité intrinsèque du logiciel, soit la *capacité fonctionnelle*, la *fiabilité*, le *rendement* et *l'utilisabilité*. Les autres facteurs du domaine de la qualité intrinsèque sont moins importants aux yeux des utilisateurs puisqu'ils s'attardent aux aspects internes du logiciel. Quant au domaine de la qualité extrinsèque, les utilisateurs sont évidemment concernés par la *valeur de l'information* issue du logiciel ainsi qu'au *gain de productivité* potentiel découlant de l'utilisation du logiciel.

La Figure 8.3 présente le modèle de qualité adapté à la vue des utilisateurs alors que le Tableau 8.1 présente la description de chacun des attributs du modèle.

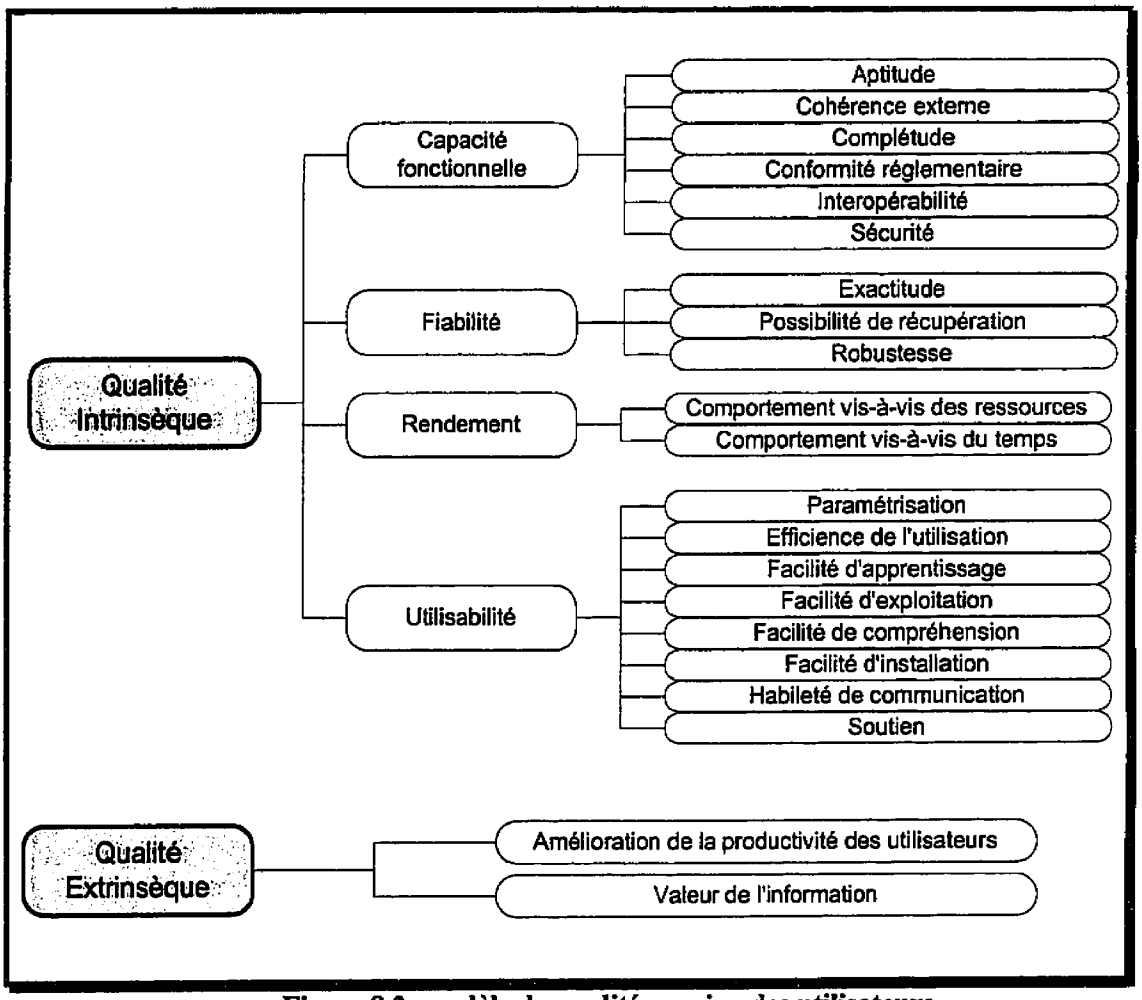


Figure 8.3: modèle de qualité, version des utilisateurs

Tableau 8.1: description des attributs liés à la vue des utilisateurs

Capacité fonctionnelle		*Ensemble de critères de qualité portant sur l'existence d'un ensemble de fonctions et leurs propriétés données. Les fonctions sont celles qui satisfont aux besoins exprimés ou implicites*
	Aptitude	Degré selon lequel le logiciel remplit les objectifs de la mission du client
	Cohérence externe	Degré de correspondance entre le contenu du logiciel d'une part, et les spécifications et la documentation d'autre part
	Complétude	Degré selon lequel toutes les composantes essentielles du logiciel sont présentes et dont chacune est entièrement développée
	Conformité réglementaire	Degré selon lequel le logiciel respecte les politiques de fonctionnement établies dans l'entreprise
	Interopérabilité	Capacité du logiciel à interagir avec d'autres systèmes donnés
	Sécurité	Aptitude du logiciel à empêcher tout accès non autorisé (accidentel ou délibéré) à ses composantes
Fiabilité		*Ensemble de critères de qualité portant sur l'aptitude du logiciel à maintenir son niveau de service dans des conditions précises et pendant une période déterminée*
	Exactitude	Degré selon lequel les sorties du logiciel sont suffisamment précises pour satisfaire l'utilisation prévue
	Possibilité de récupération	Capacité du logiciel de rétablir son niveau de service et de restaurer les informations directement affectées en cas de défaillance, et sur le temps et l'effort nécessaires pour le faire
	Robustesse	Aptitude du logiciel à maintenir un niveau de service donné en cas de défaut du logiciel ou de violation des hypothèses de ses spécifications (de son interface)
Rendement		*Ensemble de critères de qualité portant sur le rapport existant entre le niveau de service d'un logiciel et la quantité de ressources utilisées, dans des conditions déterminées*
	Comportement vis-à-vis des ressources	Respect des contraintes de ressources lors de l'exécution du logiciel: quantité de ressources utilisées et durée de leur utilisation
	Comportement vis-à-vis du temps	Respect des contraintes temporelles lors de l'exécution du logiciel: temps de réponse, temps de traitement et débit
Utilisabilité		*Ensemble de critères de qualité portant sur l'effort nécessaire pour l'utilisation du logiciel et sur l'évaluation individuelle de cette utilisation par un*

		ensemble défini ou implicite d'utilisateurs
	Efficience de l'utilisation	Degré selon lequel les tâches du logiciel sont regroupées en module logique
	Facilité d'apprentissage	Effort que doit faire l'utilisateur pour apprendre l'application du logiciel
	Facilité d'exploitation	Effort que doit faire l'utilisateur pour exploiter et contrôler l'application du logiciel
	Facilité de compréhension	Effort que doit faire l'utilisateur pour reconnaître la logique du logiciel et sa mise en œuvre
	Facilité d'installation	Effort requis pour installer le logiciel ou des composantes du logiciel ainsi que l'effort requis pour la réinstallation (pour les versions subséquentes)
	Habileté de communication	Degré selon lequel le logiciel est conçu en accord avec les caractéristiques psychologiques et physiologiques des utilisateurs
	Paramétrisation	Degré selon lequel le logiciel ou des composantes du logiciel peuvent être adaptés à différents contextes d'utilisation
	Soutien	Le degré selon lequel le logiciel assiste et guide l'utilisateur
Valeur de l'information		*Bénéfices attribuables à la valeur de l'information produite par le logiciel et au soutien que cette information apporte à la prise de décision*
Amélioration de la productivité des utilisateurs		*Bénéfices attribuables à l'amélioration de la productivité des utilisateurs*

8.4.2 Vue des réalisateurs

Les réalisateurs, quant à eux, sont intéressés par la qualité du produit dans son état final, mais aussi par la qualité dans des états intermédiaires. De ce fait, les aspects internes du logiciel tels la *maniabilité*, *l'auditabilité* sont d'une grande importance. La *maniabilité* est un nouveau facteur de qualité que nous avons élaboré. Ce facteur est le regroupement de trois autres facteurs qui sont fortement chevauchés et qui par le fait

même portent souvent à confusion; ces facteurs sont la *portabilité*, la *réutilisabilité* et la *maintenabilité*. En ce sens, lorsqu'un logiciel se voit ajouter des nouvelles caractéristiques, est-ce que l'ancien logiciel est maintenu pour ajouter des nouvelles caractéristiques ou l'ancien logiciel est-il réutilisé pour produire un nouveau logiciel ? La même remarque peut être faite pour la *portabilité* qui se résume à ajouter des nouvelles caractéristiques pour rendre le logiciel opérationnel sur une autre plate-forme. Notre objectif avec l'attribut *maniabilité* est d'éliminer la confusion qui règne avec la considération des trois autres attributs en parallèle. La *maniabilité* est donc un attribut plus général qui englobe la *portabilité*, la *réutilisabilité* et la *maintenabilité*.

Les réalisateurs sont également intéressés par d'autres attributs relevant des aspects internes du logiciel soit la *cohérence*, la *compréhensibilité*, la *concision*, la *modularité* et la *standardisation*. Ces critères de qualité ont été regroupés sous un facteur de qualité appelé *clarté de l'architecture interne*.

La Figure 8.4 présente le modèle de qualité adapté à la vue des utilisateurs alors que le Tableau 8.2 présente la description de chacun des attributs du modèle.

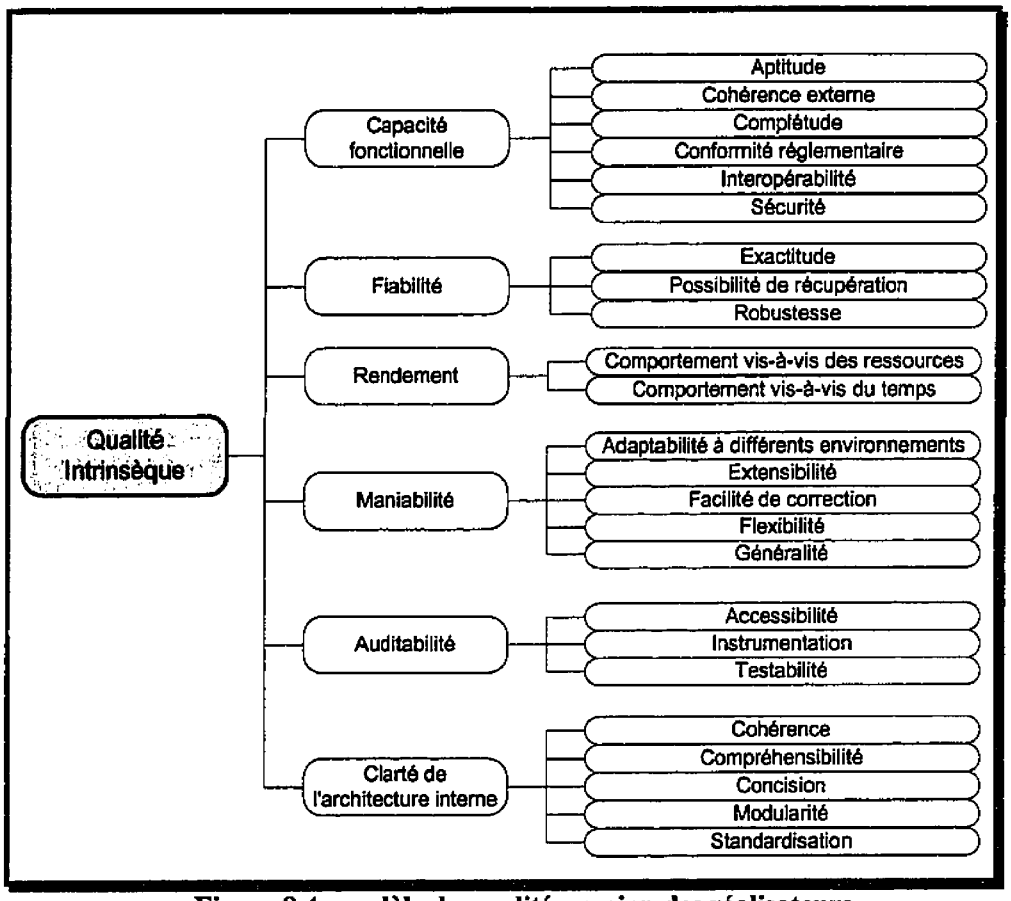


Figure 8.4: modèle de qualité, version des réalisateurs

Tableau 8.2: description des attributs de la vue des réalisateurs

[illegible]		[illegible]
Clarté de l'architecture interne		*Ensemble de critères de qualité ayant un impact sur la clarté de l'architecture interne du logiciel*
	Cohérence	Degré d'uniformité des composantes du logiciel dans leur notation, leur terminologie et leur symbolisme
	Compréhensibilité	Niveau de clarté et d'intelligibilité des composantes du logiciel facilitant la compréhension et le retraçage mental des informations, telles la logique des éléments, l'interaction entre les différentes composantes, etc.
	Concision	Degré selon lequel les composantes du logiciel ne contiennent que les informations nécessaires et suffisantes

		(absence d'excès d'informations)
	Modularité	Degré selon lequel les composantes du logiciel sont hautement cohésives tout en étant faiblement couplées
	Standardisation	Niveau de respect des composantes du logiciel vis-à-vis les standards adoptés
Capacité fonctionnelle		*Ensemble de critères de qualité portant sur l'existence d'un ensemble de fonctions et leurs propriétés données. Les fonctions sont celles qui satisfont aux besoins exprimés ou implicites*
	Aptitude	Degré selon lequel le logiciel satisfait ses spécifications
	Cohérence externe	Degré de correspondance entre le contenu du logiciel d'une part, et les spécifications et la documentation d'autre part
	Complétude	Degré selon lequel toutes les composantes essentielles du logiciel sont présentes et dont chacune est entièrement développée
	Conformité réglementaire	Degré selon lequel le logiciel respecte l'application des normes, des conventions, des réglementations de droit ou des prescriptions similaires
	Interopérabilité	Capacité du logiciel à interagir avec d'autres systèmes donnés
	Sécurité	Aptitude du logiciel à empêcher tout accès non autorisé (accidentel ou délibéré) à ses composantes
Fiabilité		*Ensemble de critères de qualité portant sur l'aptitude du logiciel à maintenir son niveau de service dans des conditions précises et pendant une période déterminée*
	Exactitude	Degré selon lequel la précision des extrants du logiciel correspond à ce qui est attendu dans des conditions données
	Possibilité de récupération	Capacité du logiciel de rétablir son niveau de service et de restaurer les informations directement affectées en cas de défaillance, et sur le temps et l'effort nécessaires pour le faire
	Robustesse	Aptitude du logiciel à maintenir un niveau de service donné en cas de défaut du logiciel ou de violation des hypothèses de ses spécifications (de son interface)
Maniabilité		*Ensemble de critères de qualité portant sur l'effort nécessaire pour modifier le comportement du logiciel ou des composantes du logiciel*
	Adaptabilité à différents environnements	Possibilité d'adaptation du logiciel ou des composantes du logiciel à différents environnements (matériel et logiciel) sans que l'on ait recours à d'autres actions ou moyens que ceux prévus à cet effet
	Extensibilité	Effort requis pour étendre la capacité ou la performance du logiciel ou des composantes du logiciel

	Facilité de correction	Effort requis pour localiser, diagnostiquer et corriger les fautes dans le logiciel ou dans les composantes du logiciel
	Flexibilité	Effort requis pour modifier le logiciel ou des composantes du logiciel pour accommoder des changements dans les spécifications
	Généralité	Utilité du logiciel ou des composantes du logiciel au delà des spécifications initiales
Rendement		*Ensemble de critères de qualité portant sur le rapport existant entre le niveau de service d'un logiciel et la quantité de ressources utilisées, dans des conditions déterminées*
	Comportement vis-à-vis des ressources	Respect des contraintes de ressources lors de l'exécution du logiciel: quantité de ressources utilisées et durée de leur utilisation
	Comportement vis-à-vis du temps	Respect des contraintes temporelles lors de l'exécution du logiciel: temps de réponse, temps de traitement et débit
Auditabilité		*Ensemble de critères de qualité portant sur l'effort pour vérifier l'opération et la performance du logiciel ou des composantes du logiciel*
	Accessibilité	Degré selon lequel le logiciel facilite l'utilisation sélective de ses différentes composantes
	Instrumentation	Degré selon lequel le logiciel surveille sa propre utilisation (son statut) et identifie les erreurs qui surviennent
	Testabilité	Effort requis pour vérifier l'opération du logiciel ainsi que ses performances

8.4.3 Vue du maître d'ouvrage

Le maître d'ouvrage s'intéresse évidemment à la qualité intrinsèque du logiciel mais d'un point de vue global. Toutefois, ce dernier est particulièrement concerné par tous les aspects de la qualité extrinsèque, aspects lui permettant de mesurer l'impact du logiciel à plusieurs égards.

La Figure 8.5 présente le modèle de qualité adapté à la vue du maître d'ouvrage alors que le Tableau 8.3 présente la description de chacun des attributs du modèle.

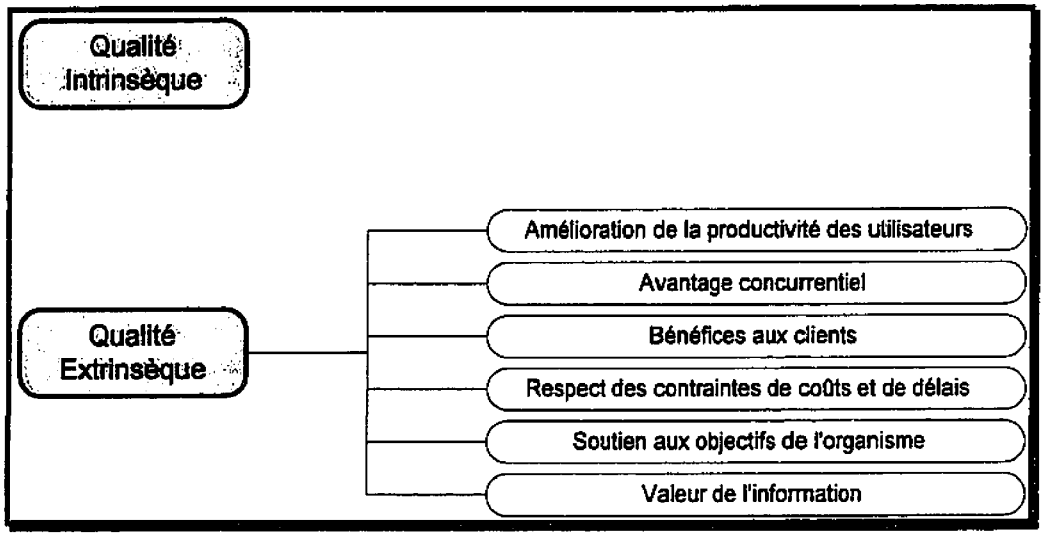


Figure 8.5: modèle de qualité, version du maître d'ouvrage

Tableau 8.3: description des attributs de la vue du maître d'ouvrage

Amélioration de la productivité des utilisateurs	*Bénéfices attribuables à l'amélioration de la productivité des utilisateurs résultant en une baisse des coûts par unité de travail*
Avantage concurrentiel	*Bénéfices attribuables à l'amélioration de la position stratégique de l'organisme*
Bénéfices aux clients	*Bénéfices attribuables à l'augmentation de la satisfaction des clients face aux services et/ou produits de l'entreprise*
Respect des contraintes de coûts et de délais	*Bénéfices attribuables aux respect des contraintes de développement à la fois en terme de coûts et de temps de développement*
Soutien aux objectifs de l'organisme	*Bénéfices attribuables à l'alignement des objectifs du logiciel avec ceux de l'organisme*
Valeur de l'information	*Bénéfices attribuables à la valeur de l'information produite par le logiciel et au soutien que cette information apporte à la prise de décision*

8.5 Le choix des attributs internes

Ayant choisi les attributs externes, la prochaine étape de l'élaboration est de choisir les attributs internes, communément appelés métriques logicielles.

Malheureusement, cette étape a été annulée puisqu'elle entre en conflit avec les objectifs de notre modèle de qualité. Voulant développer un modèle à portée générale, il s'est avéré impossible de trouver des métriques s'appliquant à toutes les classes d'applications logicielles. Ainsi, les métriques sont à caractères très spécifiques et ne peuvent être utilisées que pour une seule classe d'application logicielle.

Un autre fait important vient appuyer notre décision : il s'agit de la fébrilité du domaine des métriques logicielles. Ainsi, selon l'avis de plusieurs auteurs, les métriques existantes sont non-intuitives, non instructives et manquent une base fondamentale pour comprendre leur implication comparée à d'autres mesures. Cette opinion est partagée par ISO-9126 (1991) qui n'a pas inclus de métriques dans son modèle prétextant que l'état de l'art dans ce domaine n'est pas encore assez avancé.

8.6 Le choix de la procédure d'agrégation de données

L'architecture logicielle ainsi que les attributs peuplant le modèle ayant été choisis, il reste à choisir une procédure pour agréger les valeurs des critères de qualité jusqu'aux attributs de haut niveau. Notre choix s'est arrêté sur le processus d'analyse hiérarchique développé par Saaty (1990), procédure qui est présentée au chapitre 7. Plusieurs raisons expliquent ce choix. Premièrement, une étude réalisée par Schoemaker et Waid (1982) a démontrée que cette procédure était la meilleure parmi un ensemble de procédures d'agrégation. Deuxièmement, la procédure est compatible

avec l'architecture en *N arborescences* qui a été choisie. Finalement, la procédure possède la majorité des qualités désirables d'une méthode d'agrégation qui ont été définies au chapitre 7. Ainsi, la méthode propage d'une façon cohérente les données des niveaux inférieurs vers les niveaux supérieurs tout en respectant la théorie de la mesure. Elle peut gérer simultanément des données quantitatives et qualitatives et également affecter des priorités entre les attributs.

8.7 Notre modèle en action

L'élaboration du modèle étant terminée, il serait intéressant de l'utiliser dans un cas réel pour noter comment il se comporte. Ainsi, dans cette section, nous évaluons la qualité d'un logiciel en se servant du modèle. Le logiciel évalué est une application du domaine de la simulation qui a été développée en langage orienté-objet par une équipe de génie logiciel.

L'évaluation a été conduite par l'équipe de développement permettant ainsi de dresser un portrait de la vue des réalisateurs. Ces derniers ont participé à une séance de discussion ayant pour but de fournir les données nécessaires à l'évaluation du logiciel. Nous allons donc, au cours des prochains paragraphes, présenter l'analyse de ces données. Cette analyse prend en considération la méthode d'agrégation que nous avons choisie.

Ayant en main la version des réalisateurs du modèle, l'équipe de développement avait comme premier objectif de déterminer l'importance des attributs du modèle. Pour cela, des comparaisons entre paires d'attributs ont eu lieu. Ces comparaisons furent basées sur l'échelle d'évaluation proposée par Saaty (1990), échelle supposée optimale (voir chapitre 7). Les matrices qui suivent présentent l'évaluation de l'importance des

attributs selon leur niveau dans le modèle. La première matrice (Tableau 8.4) s'intéresse aux facteurs de qualité, alors que les autres matrices (Tableau 8.5 au Tableau 8.10) se préoccupent des critères de qualité associés à chacun des facteurs.

Tableau 8.4 : importance relative des facteurs de qualité

	1	1/3	1/3	1/5	1/3	1/5
	3	1	1	1	5	1
	3	1	1	1/3	1	1/3
	5	1	3	1	5	1/3
	3	1/5	1	1/5	1	1/7
	5	1	3	3	7	1

Tableau 8.5 : importance relative des critères de qualité liés au facteur *auditabilité*

	1	1/5	1/7
	5	1	1/3
	7	3	1

Tableau 8.6 : importance relative des critères de qualité liés au facteur *capacité fonctionnelle*

	1	1	7	5	7	9
	1	1	5	7	7	9
	1/7	1/5	1	3	5	9
	1/5	1/7	1/3	1	5	7
	1/7	1/7	1/5	1/5	1	1/3
	1/9	1/9	1/9	1/7	3	1

Tableau 8.7 : importance relative des critères de qualité liés au facteur *clarté de l'architecture interne*

	1	1/3	1/5	1/7	1
	3	1	3	1	3
	5	1/3	1	1/3	3
	7	1	3	1	5
	1	1/3	1/3	1/5	1

Tableau 8.8 : importance relative des critères de qualité liés au facteur *fiabilité*

	1	7	7
	1/7	1	1/3
	1/7	3	1

Tableau 8.9 : importance relative des critères de qualité liés au facteur *maniabilité*

	1	1/5	1/7	1/5	1/3
	5	1	1/3	1/5	5
	7	3	1	5	5
	5	5	1/5	1	5
	3	1/5	1/5	1/5	1

Tableau 8.10 : importance relative des critères de qualité liés au facteur *rendement*

	1	1/5
	5	1

Chaque matrice génère un vecteur propre qui classifie l'importance des attributs à chaque niveau du modèle. En utilisant les vecteurs propres résultants comme facteur de pondération, chaque attribut du logiciel est pondéré. Les résultats sont agrégés niveau par niveau pour obtenir une évaluation globale du logiciel.

Les calculs des vecteurs propres associés à chacune des matrices exprimant l'importance relative des attributs du modèle (Tableau 8.4 au Tableau 8.10) sont présentés du Tableau 8.11 au Tableau 8.17.

Tableau 8.11 : vecteurs propres des facteurs de qualité

	(1)/[20]	(1/3)/[68/15]	(1/3) /[28/3]	(1/5)/[86/15]	(1/3) /[58/3]	(1/5) /[316/105]	0,05
	(3) /[20]	(1) /[68/15]	(1) /[28/3]	(1) /[86/15]	(5) /[58/3]	(1) /[316/105]	0,21
	(3) /[20]	(1) /[68/15]	(1) /[28/3]	(1/3) /[86/15]	(1) /[58/3]	(1/3) /[316/105]	0,12
	(5) /[20]	(1) /[68/15]	(3) /[28/3]	(1) /[86/15]	(5) /[58/3]	(1/3) /[316/105]	0,22
	(3) /[20]	(1/5) /[68/15]	(1) /[28/3]	(1/5) /[86/15]	(1) /[58/3]	(1/7) /[316/105]	0,07
	(5) /[20]	(1) /[68/15]	(3) /[28/3]	(3) /[86/15]	(7) /[58/3]	(1) /[316/105]	0,33
	20	68/15	28/3	86/15	58/3	316/105	

Tableau 8.12 : vecteurs propres des critères de qualité associés au facteur *auditabilité*

	(1)/[13]	(1/5) / [21/5]	(1/7) /[41/21]	0,07
	(5) /[13]	(1) /[21/5]	(1/3) /[41/21]	0,28
	(7) /[13]	(3) /[21/5]	(1) /[41/21]	0,64
	13	21/5	41/21	

Tableau 8.13 : vecteurs propres des critères de qualité associés au facteur *capacité fonctionnelle*

	(1) /[818/315]	(1) /[818/315]	(7) /[614/45]	(5) /[572/35]	(7) /[28]	(9)/[106/3]	0,35
	(1) /[818/315]	(1) /[818/315]	(5) /[614/45]	(7) /[572/35]	(7) /[28]	(9)/[106/3]	0,34
	(1/7) /[818/315]	(1/5) /[818/315]	(1) /[614/45]	(3) /[572/35]	(5) /[28]	(9)/[106/3]	0,14
	(1/5) /[818/315]	(1/7) /[818/315]	(1/3) /[614/45]	(1) /[572/35]	(5) /[28]	(7)/[106/3]	0,10
	(1/7) /[818/315]	(1/7) /[818/315]	(1/5) /[614/45]	(1/5) /[572/35]	(1) /[28]	(1/3)/[106/3]	0,03
	(1/9) /[818/315]	(1/9) /[818/315]	(1/9) /[614/45]	(1/7) /[572/35]	(3) /[28]	(1)/[106/3]	0,04
	818/315	818/315	614/45	572/35	28	106/3	

Tableau 8.14 : vecteurs propres des critères de qualité associés au facteur *clarté de l'architecture interne*

	(1)/[17]	(1/3) /[9/3]	(1/5) /[113/15]	(1/7) /[281/105]	(1) /[13]	0,07
	(3) /[17]	(1) /[9/3]	(3) /[113/15]	(1) /[281/105]	(3) /[13]	0,30
	(5) /[17]	(1/3) /[9/3]	(1) /[113/15]	(1/3) /[281/105]	(3) /[13]	0,18
	(7) /[17]	(1) /[9/3]	(3) /[113/15]	(1) /[281/105]	(5) /[13]	0,38
	(1) /[17]	(1/3) /[9/3]	(1/3) /[113/15]	(1/5) /[281/105]	(1) /[13]	0,07
	17	9/3	113/15	281/105	13	

Tableau 8.15 : vecteurs propres des critères de qualité associés au facteur *fiabilité*

	(1) /[9/7]	(7) /[11]	(7) /[25/3]	0,75
	(1/7) /[9/7]	(1) /[11]	(1/3) /[25/3]	0,08
	(1/7) /[9/7]	(3) /[11]	(1) /[25/3]	0,17
	9/7	11	25/3	

Tableau 8.16 : vecteurs propres des critères de qualité associés au facteur maniabilité

	(1) /[21]	(1/5) /[47/5]	(1/7) /[197/105]	(1/5) /[33/5]	(1/3) /[49/3]	0,04
	(5) /[21]	(1) /[47/5]	(1/3) /[197/105]	(1/5) /[33/5]	(5) /[49/3]	0,17
	(7) /[21]	(3) /[47/5]	(1) /[197/105]	(5) /[33/5]	(5) /[49/3]	0,45
	(5) /[21]	(5) /[47/5]	(1/5) /[197/105]	(1) /[33/5]	(5) /[49/3]	0,27
	(3) /[21]	(1/5) /[47/5]	(1/5) /[197/105]	(1/5) /[33/5]	(1) /[49/3]	0,07
	21	47/5	197/105	33/5	49/3	

Tableau 8.17: vecteurs propres des critères de qualité associés au facteur *rendement*

	(1) /[6]	(1/5)/[6/5]	0,17
	(5) /[6]	(1) /[6/5]	0,83
	6	6/5	

Il est à noter que les vecteurs propres associés à l'échelle d'évaluation doivent aussi être calculés, calculs qui sont présentés au Tableau 8.18.

Tableau 8.18: vecteurs propres associés à l'échelle d'évaluation

	(1) /[25]	(1/3) /[49/3]	(1/5) /[143/15]	(1/7) /[491/105]	(1/9) /[1689/945]	0,03
	(3) /[25]	(1) /[49/3]	(1/3) /[143/15]	(1/5) /[491/105]	(1/7) [1689/945]	0,07
	(5) /[25]	(3) /[49/3]	(1) /[143/15]	(1/3) /[491/105]	(1/5) [1689/945]	0,13
	(7) /[25]	(5) /[49/3]	(3) /[143/15]	(1) /[491/105]	(1/3) [1689/945]	0,26
	(9) /[25]	(7) /[49/3]	(5) /[143/15]	(3) /[491/105]	(1) [1689/945]	0,50
	25	49/3	143/15	491/105	1689/945	

Les calculs relatifs au degré d'importance des attributs du modèle étant terminés, il est possible de s'attarder à l'évaluation même de ces attributs. Pour ce faire, les membres de l'équipe de développement ont évalué chacun des critères de qualité en utilisant toujours l'échelle d'évaluation proposée par Saaty (1990). Cette évaluation est présentée à la Figure 8.6. Il est à noter que seulement les attributs du dernier niveau du modèle doivent être évalués, l'évaluation des autres attributs étant dictée par la méthode d'agrégation.

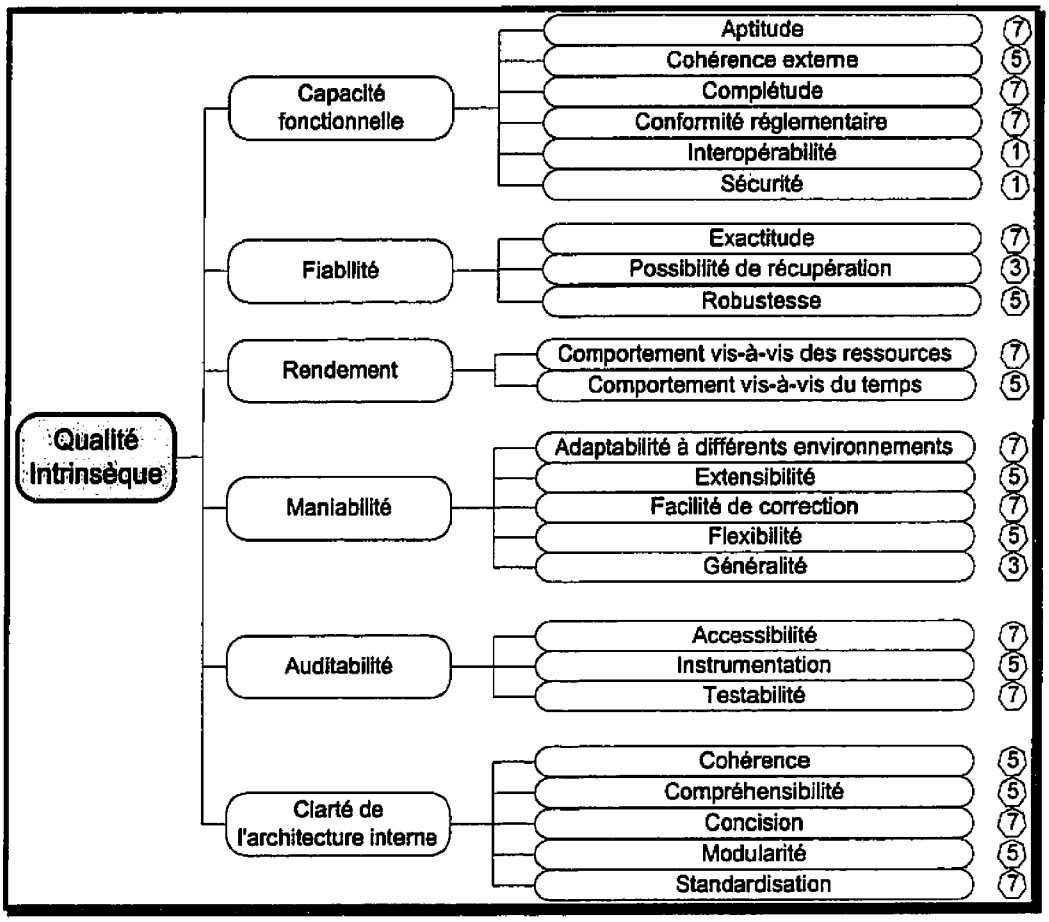


Figure 8.6: évaluation des critères de qualité

À ce stade, nous avons en main toutes les données nécessaires pour être en mesure de dresser une évaluation globale du simulateur.

Pour cela, les vecteurs propres de chacun des critères de qualité sont pondérés avec les évaluations qui leurs sont assignées. La pondération est basée sur les vecteurs propres de chacun des niveaux de l'échelle d'évaluation. La somme des vecteurs propres

associés à un facteur de qualité correspond à l'évaluation non normalisée de ce facteur. Les calculs sont présentés du Tableau 8.19 au Tableau 8.24.

Tableau 8.19: pondération des vecteurs propres associés au facteur *auditabilité*

	0,07	0,26	0,02
	0,28	0,13	0,04
	0,64	0,26	0,17
		Total	0,23

Tableau 8.20: pondération des vecteurs propres associés au facteur *capacité fonctionnelle*

	0,35	0,26	0,09
	0,34	0,13	0,04
	0,14	0,26	0,04
	0,10	0,26	0,03
	0,03	0,03	0,0
	0,04	0,03	0,0
		total	0,2

Tableau 8.21: pondération des vecteurs propres associés au facteur *clarté de l'architecture interne*

	0,07	0,13	0,01
	0,30	0,13	0,04
	0,18	0,26	0,05
	0,38	0,13	0,05
	0,07	0,26	0,02
		total	0,17

Tableau 8.22: pondération des vecteurs propres associés au facteur *fiabilité*

	0,75	0,26	0,2
	0,08	0,07	0,01
	0,17	0,13	0,02
		total	0,23

Tableau 8.23: pondération des vecteurs propres associés au facteur *maniabilité*

	0,04	0,26	0,01
	0,17	0,13	0,02
	0,45	0,26	0,12
	0,27	0,13	0,04
	0,07	0,07	0,0
		total	0,19

Tableau 8.24: pondération des vecteurs propres associés au facteur *rendement*

	0,17	0,26	0,04
	0,83	0,13	0,11
		total	0,15

Enfin, pour obtenir l'évaluation globale, les vecteurs propres pondérés de chacun des facteurs de qualité sont disposés dans une matrice qui est multipliée par la matrice qui exprime l'importance de chacun des facteurs. Le résultat du produit matriciel est présenté à la Figure 8.7.

$$\begin{bmatrix} 0,23 \\ 0,2 \\ 0,17 \\ 0,23 \\ 0,19 \\ 0,15 \end{bmatrix} \otimes \begin{bmatrix} 0,05 & 0,21 & 0,12 & 0,22 & 0,07 & 0,33 \end{bmatrix} = 0,18$$

Figure 8.7: calculs liés à l'évaluation globale non normalisée

Finalement, le résultat obtenu par la multiplication matricielle est normalisé par l'évaluation maximale. Cette dernière correspond simplement au vecteur propre associé au dernier niveau de l'échelle d'évaluation, soit 0,5.

L'évaluation globale est :

$$\frac{0,18}{0,5} = 0,36 = 36\%$$

Nous pouvons cependant nous poser certaines questions en ce qui concerne la signification d'une telle évaluation. Jusqu'à quel point peut-on s'y fier ? L'évaluation est-elle trop précise en regard de l'état de l'art dans le domaine de l'évaluation de la qualité du logiciel ? Quel est l'impact d'une telle évaluation sur l'équipe de développement ? Tant de questions qui restent à ce jour sans réponse...

CONCLUSION

Au dire de Garvin [G84], la qualité est un concept complexe présentant plusieurs facettes. C'est également la source de beaucoup de confusion : les gestionnaires échouent souvent à communiquer précisément le fond de leur pensée lorsqu'ils utilisent le terme qualité. Le résultat tourne souvent en des débats interminables et une inhabileté à démontrer de réels progrès au niveau de la qualité.

Ce document a présenté un modèle de qualité logicielle qui, nous l'espérons, contribuera à améliorer notre compréhension de ce qu'est la qualité du logiciel.

Bien que nous ne prétendions pas que le modèle de qualité logicielle que nous avons développé soit mieux que ceux existants, nous pensons qu'il évite plusieurs embûches qui ont rendu plusieurs des modèles existants inutilisables.

Avant d'entrer dans le coeur de l'ouvrage d'élaboration de notre modèle, nous avons pris soin de définir clairement quelle était sa portée. Ainsi, notre modèle en est un à caractère général et vise à être applicable à toutes les classes d'applications logicielles. La majorité des modèles existants n'effleurent pas ce point, ce qui nous laisse indécis quant au contexte d'utilisation de ces modèles.

Étant en accord avec les auteurs qui prétendent que la qualité logicielle dépend du point de vue qui est considéré, nous avons développé une version du modèle pour chacune des trois vues de la qualité reconnues par ISO-9126 (1991). Il y a donc une version pour les utilisateurs, une pour les réalisateurs et une dernière pour le maître d'ouvrage.

À notre connaissance, aucun des modèles existants n'a suivi une procédure d'élaboration aussi rigoureuse que la nôtre. Notre procédure est décomposée en quatre étapes principales.

La première étape consiste à choisir l'architecture logicielle. Ayant défini plusieurs critères de sélection, la famille des architectures *N arborescences* fut la seule qui répondait à tous les critères. Sommairement, cette architecture logicielle permet d'accommoder toute la connaissance relative à la qualité logicielle et cela, d'une façon constructive, raffinable et intellectuellement gérable.

La deuxième étape est de choisir les attributs externes du modèle. Treize facteurs de qualité regroupés dans deux domaines distincts furent retenus. De ces facteurs découlent trente-deux critères de qualité. L'ensemble d'attributs retenus est, selon nous, un ensemble complet, non entrelacé et compatible. Malheureusement, la troisième étape fut annulée puisqu'elle entre en conflit avec les objectifs de notre modèle de qualité. Voulant développer un modèle à portée générale, il s'est avéré impossible de trouver des métriques s'appliquant à toutes les classes d'applications logicielles.

La dernière étape fut le choix de la procédure d'agrégation de données, procédure qui permet d'utiliser notre modèle. Notre choix s'est arrêté sur le processus d'analyse hiérarchique. Cette procédure est compatible avec notre architecture logicielle et possède la majorité des qualités que nous recherchions.

Notre procédure d'élaboration soulève deux points qui sont très souvent négligés dans l'élaboration des modèles existants. Il s'agit de l'architecture logicielle sur laquelle le modèle est basé ainsi que la procédure d'agrégation des données qui permet d'utiliser le modèle. Bien que certains modèles aient abordé d'une façon relativement rigoureuse le

sujet de la procédure d'agrégation des données, la plupart des modèles négligent ce point. Un tel oubli rend le modèle pratiquement inutilisable. Pour ce qui est de l'architecture logicielle, aucun de la dizaine des modèles étudiés n'abordent ce point. Ainsi, il semble que les modèles existants soient basés sur des architectures plutôt douteuses. Il ne semble pas y avoir eu d'étude justifiant le choix d'une architecture logicielle plutôt qu'une autre.

Étant dans une ère de qualité totale, il est important que le modèle soit constamment amélioré et évolue avec l'apparition de nouvelles technologies. Idéalement, le modèle devrait être modifié en fonction de l'environnement dans lequel il est utilisé.

Pour nous convaincre du potentiel de notre modèle, nous l'avons utilisé pour évaluer la qualité d'un logiciel du domaine de la simulation. Les résultats finaux de l'évaluation furent intéressants; malheureusement, nous sommes perplexes quant à leur interprétation.

Bien que notre modèle semble être un pas dans la quête de ce qu'est la qualité logicielle, certains travaux futurs restent à réaliser.

Notre modèle étant innovateur pour ce qui est de la considération des trois vues de la qualité logicielle (vue du maître d'ouvrage, des utilisateurs et des réalisateurs), plusieurs questions sont sans réponse à cet égard. Ainsi, est-il possible que la qualité d'un logiciel soit perçue différemment par le maître d'ouvrage, les utilisateurs et les réalisateurs? Si oui, qu'est ce que cela implique?

Bien que notre modèle ait été développé avec une firme de consultant en informatique, il n'a été éprouvé qu'une seule fois en situation réelle. Il est primordial que le modèle

soit utilisé dans quelques projets pilotes pour permettre de déceler ses faiblesses et ainsi l'améliorer.

RÉFÉRENCES

ARTHUR L. J. (1993), *Improving Software Quality*, New York : John Wiley & Sons.

ALBRECHT A. J., GAFFNEY, J. E. (1983), « Software Function, Source Lines of Code, and Development Effort Prediction : A Software Science Validation, » *IEEE Transactions on Software Engineering*, IEEE Computer Society, New York, USA, vol. 9, pp. 639-648.

ARTHUR JAMES D., NANCE RICHARD E (1987), « Developping an Automated Procedure for Evaluating Software Development Methodologies Products, » *Systems Research Center*, Blacksburg, Virginia, Tech. Rep. SRC-87-007.

ARTHUR JAMES D., NANCE RICHARD E., BALCI OSMAN (1993), « Establishing Software Development Process Control : Technical Objectives, Operational Requirements, and the Foundational Framework, » *The Journal of Systems and Software, Elsevier Science Publish,* vol 22, no. 2, pp. 117-128.

ARTHUR JAMES D, NANCE RICHARD E., BUNDY G. N., DORSEY E. V., HENRY J. (1991), « Software Quality Measurement : Validation of a Foundational Approach, » *Systems Research Center*, Blacksburg, Virginia, Tech. Rep. SRC-91-002.

BERGE CLAUDE (1970), *Graphes et hypergraphes*, Monographies universitaires de mathématiques, Dunod, Paris.

BRONOWSKI, JACOB (1973), *The Ascent of Man,* Little, Brown and Co, Boston.

BOWEN T. P (1976), « Specification of Software Quality Attributes, » *Rome Laboratory*, Tech. Rep. RADC-TR-85-3, vol. 1, no. 3, New York.

BOEHM B. (1981), *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ.

BAILEY D. H. (1991), « Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers, Supercomputer, » vol 4, no. 7.

BASILI VICTOR R. (1992), « Software Modeling and Measurement : The Goal/Question/Metric Paradigm, » Technical Report , CS-TR-2956 UMIACS-TR-92, University of Maryland, College Park, MD.

BOEHM B. W., BROWN, J. R., KASPAR H., LIPOW M., MACLEOD G. J., MERRITT J. (1978), *Characteristics of Software Quality, Elsevier*, New York, North-Holland.

BOEHM B. W., BROWN J. R., LIPOW M. (1976), « Quantitative Evaluation of Software Quality, » *International Conference on Software Engineering*, pp. 592-605.

BANKER R. S., DATAR S., KEMERER C. (1991), « A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects, » *Manag. Sci.*, vol. 37.

BOLLINGER, TERRY B., MCGOWAN CLEMENT (1991), « A Critical Look at Software Capability Evaluations, » *IEEE Software*, IEEE Computer Society, New York, USA, pp. 25-41.

BASILI VICTOR R., ROMBACH HANS DIETER (1988), « The TAME Project : Towards Improvement-Oriented Software Environment, » *IEEE Transactions on Software Engineering*, IEEE Computer Society, New York, USA, vol. 14, pp. 759-773.

BOLOIX GERMINAL, ROBILLARD PIERRE N. (1994A), « Comprehensive Software Metrics Framework, » *Technical Report Polytechnique*, Les Éditions de l'École Polytechnique, Montréal, Canada.

BOLOIX GERMINAL, ROBILLARD PIERRE N. (1994B), « A Hierarchical Approach to Aggregating Software Systems Evaluation Data, » *Technical Report Polytechnique*, Les Éditions de l'École Polytechnique, Montréal, Canada.

BOLOIX GERMINAL, ROBILLARD PIERRE N. (1995A), « Project Management's Perceptions on Software Systems Evaluation, » *Technical Report Polytechnique*, Les Éditions de l'École Polytechnique, Montréal, Canada.

BOLOIX GERMINAL, ROBILLARD PIERRE N. (1995B), « A Software System Evaluation Framework, » *Technical Report Polytechnique*, Les Éditions de l'École Polytechnique, Montréal, Canada.

BOWEN T. P., WIGLE G. B., TSAI J. (1984), « Specification of software quality attributes, » *US Rome Air Development Center*, Reports D182-11678-1,D1, vol. 1, no. 3.

BASILI VICTOR R., ZELKOWITZ MARVIN V. (1978), « Analyzing Medium-Scale Software Development, » *3rd. Inter. Conference on Software Engineering*, Atlanta, Georgia, USA.

CONOVER W. J. (1971), *Practical Nonparametric Statistics*, John Wiley & Sons, New York.

CROSBY P. (1979), *Quality is Free*, McGraw-Hill, New York.

CONTE S. D., DUNSMORE H. E., SHEN V. Y. (1986), *Software Engineering Metrics and Models*, Benjamin/Cummings, Menlo Park, California.

CAVANO J. P., MCCALL J. A. (1978), « A Framework for the Measurement of Software Quality, » *Proc. ACM Software Quality Assurance Workshop*, pp. 133-139.

DROMEY R. GEOFF (1995), « A Model for Software Product Quality, » IEEE *Transactions on Software Engineering*, IEEE Computer Society, New York, USA, vol. 21, no. 2, pp. 146-162.

DUNN R., ULLMAN R. (1982), *Quality assurance for computer software*, McGraw-Hill, New York, pp. 262-263.

DEUTSCH M., WILLIS RONALD R. (1988), *Software Quality Engineering*, Prentice-Hall, , Englewood Cliffs, NJ.

FENTON NORMAND (1991), *Software Metrics : A Rigourous Approach*, Chapman & Hall.

FINNIE GAVIN R., WITTIG GERHARD E., PETKOV DONCHO I. (1993), « Prioritizing Software Development Productivity Factors Using the Analytic Hierarchy Process, » *The Journal of Systems and Software*, Elsevier Science Publishing Co. Inc., New York, USA, vol. 22, no. 2, pp. 129-139.

GUNNING ROBERT (1962), « How to Take the Fog Out of Writing, » Dartuell Press, Inc., Chicago, Ill.

GIBBONS J. D. (1971), *Nonparametric Statistical Inference*, McGraw-Hill, New York.

GARVIN D. A. (1984), « What does "product quality" really mean, » *Sloan Management Review Fall*, vol. 25.

GILB T. (1986), *Tools for Design by Objectives*, ANDERSON T. (Ed.) : "Sof. Requirements specification & testing.

GILB T. (1987), *Principles of Software engineering Management*, Addison Wesley.

GRADY ROBERT B., CASWELL D. C. (1987), *Software Metrics : Establishing a Company Wide Program*, Prentice Hall, Englewood Cliffs, NJ.

GONDRAN MICHEL, MINOUX MICHEL (1985), *Graphes et algorithmes (2e edition)*, Editions Eyrolles, Paris, France.

HALSTEAD M. H. (1977), *Elements of software science*, Elsevier, New York, North-Holland.

HUMPHREY WATTS S. (1988), « Characterizing the Software Process : A Maturity Framework, » *IEEE Software*, IEEE Computer Society, New York, USA, pp. 73-79.

HUMPHREY WATTS S. (1989), *Managing the Software Process*, Addison Wesley.

HUMPHREY WATTS S., SWEET W. L. (1987), « A Method for Assesing the Software Engineering Capability of contractors, » *Software Engineering Institute*, Carnegie Mellon University, Pittsburgh, Pennsylvania, ConCMU/SEI-87-TR-23.

HUMPHREY WATTS S., SNYDER TERRY R., WILLIS RONALD R. (1991), « Software Process Improvement at Hughes Aircraft, » *IEEE Software*, IEEE Computer Society, New York, USA, pp. 11-23.

ISO 9126 (1991) : « Critères pour l'évaluation du logiciel - Caractéristiques de qualité de logiciel et directives pour leur utilisation, » ISO.

KITCHENHAM B. (1987), « Towards a constructive quality model Part I, » *Software quality modelling, measurement and prediction, Software Engineering J,* vol. 2, no. 4, pp. 105-113.

KLEINBAUM D. G., KUPPER L. L. (1978), *Applied Regression Analysis and Other Multivariable Methods*, Duxbury Press, Boston, MA.

KAPOSI AGNES, KITCHENHAM B. (1987), « The architecture of system quality, » *Software Engineering J,* vol. 2, no. 1, pp. 2-8.

KEMAYEL L., MIL, A., OUEDERNI I. (1991), « Controllable Factors for Programmer Productivity, » *The Journal of Systems and Software*, Elsevier Science Publishing Co. Inc., New-York, USA, vol. 16, pp. 151-163.

KERNIGHAN B. W., PLAUGHER P. J. (1974), *The Elements of Programming Style*, McGraw-Hill, New York.

KEARNEY J. K., SEDLMEYER R. L., THOMPSON W. B., GREY M. A., ADLER, M. A. (1986), « Software Complexity Measurement, » *Communication of the ACM*, vol. 29, p. 1044-1050.

KITCHENHAM B., WALKER J. G. (1986), « The meaning of quality, Software Engineering 86, » *Proceedings of BCS-IEE Soft. Eng 86*, Conference, South-ampton, England.

KITCHENHAM B., WOOD L. M., DAVIES S. P. (1986), « Quality factors, criteria and metrics, » *Technical Report*, ESPRIT Rep.R1.6.1, REQUEST.

LITTLEWOOD B. (1988), « Forecasting software reliability, » *Software Reliability, Modelling*, Lectures Notes in Comp. Sg and Identification.

MCCABE T. J. (1976), « Complexity measure, » *IEEE Transactions on Software Engineering*, IEEE Computer Society, New York, USA, vol. 2, no. 4, pp. 308-320.

MCCALL J. A. (1979), « An Introduction to Software Quality Metrics, » *Software Quality Management*, pp. 127-142.

MOHANTY S. N. (1979), « Software Cost Estimation : Present and Future, » *Software Practice and Experience*, vol. 11, pp. 103-121.

MCCALL J. A., RICHARDS P. K., WALTERS G. F. (1977), « Factors in Software Quality, » *Rome Air Defense Center*, Griffiss AFB, NY, Tech. Rep. RADC-TR-77-3, vol. 1, no. 3.

NANCE RICHARD E. (1992), « Software Quality indicators : An holistic Approach to Measurement, » *Proc. 4th Ann. Software Quality Workshop*, Alexandria Bay, New York.

PIRSIG R. M. (1974), *ZEN and the art of motorcycle maintenance*, Corgi Books.

PFALTZ JOHN L. (1977), *Computer Data Structures*, McGraw-Hill, New York.

PRESSMAN ROGER S. (1992), *Software Engineering : A Practionner's Approach (3e edition)*, McGraw-Hill, New York.

ROBILLARD PIERRE N. (1993), « La qualité du logiciel et son évaluation, » *ICO*, vol. 4, no. 2 et 3, pp. 40-45.

RAMSEY C. L., BASILI VICTOR R. (1989), « An Evaluation of Expert Systems for Software Engineering Management, » *IEEE Transactions on Software Engineering*, IEEE Computer Society, New York, USA, vol. 15, no. 6, pp. 747-759.

SAATY,T. (1990), *The Analytic Hierarchy Process, (2e edition)*, RWS Publications, Pittsburgh, Pennsylvania.

SCHOEMAKER P. J., WAID C. C. (1982), « An Experimental Comparison of Different Approaches to Determinig Weights in Additive Utility Models, » *Manag. Sci.*, vol. 28, pp. 182-196.

TICHY WALTER F., LUKOWICK PAUL, PRECHELT LUTZ, HEINZ ERNST A. (1995), « Experimental Evaluation in Computer Science : A Quantitative Study, » *The Journal of Systems and Software*, Elsevier Science Publishing Co. Inc., New York, USA, vol. 28, pp. 9-18.

TREMBLAY J. PAUL, SORENSON PAUL G. (1984), *An introduction to data structures with applications (2e edition)*, McGraw-Hill, New York.

WALSTON C. E., FELIX C. P. (1977), « A Method of Programming Measurement and Estimation, » *IBM System Journal*, vol. 1, pp. 54-73.

WALKER J. G., KITCHENHAM B. (1988), « The architecture of an automated quality management system, ICL, » *Tech J*, pp. 156-170.

ANNEXE I
ARCHITECTURES LOGICIELLES COURANTES

1.1. Introduction

Dans cette annexe, nous présentons des architectures de modèles de qualité qui sont des alternatives à l'architecture simple facteur-critère-métrique. Les architectures sont présentées dans l'ordre de la classification que nous avons déjà élaborée. Il est à noter que nous avons conservé la terminologie originale pour les descriptions dans un souci de rapporter le plus fidèlement possible l'essence des architectures.

1 .2. Architecture de IEEE std. 1061-1992

Graphe N-parti

L'architecture associée au standard IEEE std. 1061-1992 (1992), qui est présentée à la Figure 1 .1, est conçue pour être flexible. Au dire de ses auteurs, elle permet l'addition, la suppression et la modification des facteurs, sous-facteurs et métriques. Ces allégations sont confirmées par l'analyse du type d'architecture (graphe N-parti) auquel appartient l'architecture sous-étude. Chaque niveau peut être étendu jusqu'à plusieurs sous-niveaux.

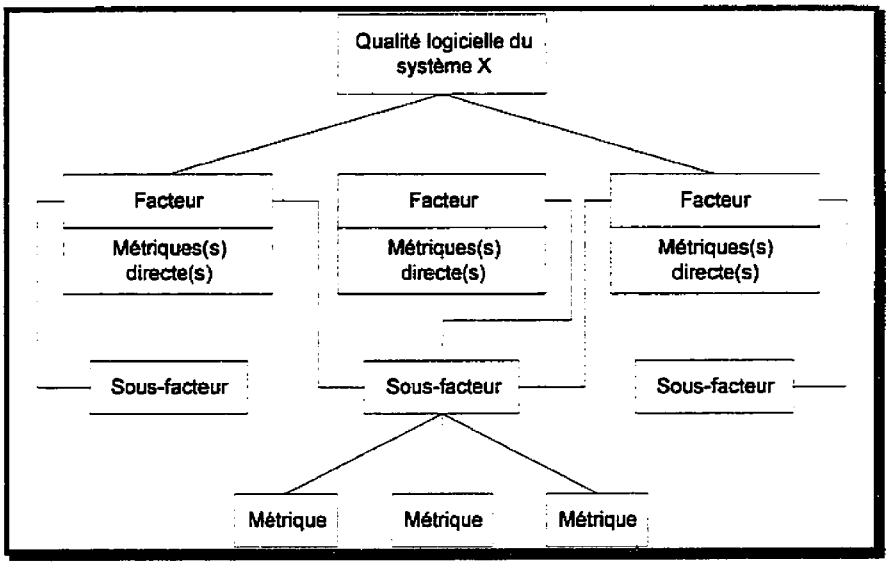


Figure 1 .1 : architecture de IEEE std. 1061-1992

La différence fondamentale entre cette architecture et celle facteur-critère-métrique est qu'une ou plusieurs métriques directes sont associées à chacun des facteurs. Ces métriques agissent à titre de représentations quantitatives du facteur de qualité. Par exemple, une métrique directe pour le facteur *fiabilité* pourrait être le *temps entre deux défaillances*. Chaque facteur devrait avoir une ou plusieurs métriques directes associées et des valeurs cibles. Sinon, il n'y a aucune façon de déterminer si le facteur a été atteint. Ces métriques directes semblent donc correspondre aux exigences de qualité pour un logiciel en particulier.

Comme pour ce qui est de l'architecture facteur-critère-métrique, au troisième niveau de la hiérarchie, les sous-facteurs sont décomposés en métriques qui sont utilisées pour mesurer non pas le produit final, mais les produits intermédiaires et les processus durant le cycle de vie du développement. Les valeurs des métriques directes ou les valeurs des facteurs sont normalement non disponibles ou dispendieuses à obtenir tôt dans le cycle de vie du logiciel. Ainsi, les métriques du troisième niveau, qui sont validées contre les métriques directes, sont utilisées pour estimer les valeurs des facteurs tôt dans le cycle de vie du logiciel.

1 .3. Architecture de Arthur, Nance et Balci

Graphe N-parti

L'architecture développée par Arthur et al. (1993) est particulièrement intéressante puisqu'elle regroupe à la fois des concepts d'assurance qualité et d'évaluation de la qualité logicielle. Plus précisément, l'architecture est dénommée par le sigle OPA ("Objectives/Principles/Attributes")

À l'aide de cette architecture, un ensemble d'*objectifs, (*qui correspondent au but au niveau du projet) peut être défini. Pour atteindre ces objectifs, il faut adhérer à certains *principes* qui caractérisent le processus selon lequel le produit est développé. En adhérant à ces principes, le produit devrait posséder des attributs considérés comme étant désirables et bénéfiques. Pour être en mesure de déterminer si un produit possède ces attributs, il faut observer les propriétés du produit. Ces propriétés sont connues sous l'appellation d'indicateurs de qualité logicielle (« SQI - software quality indicators ») qui correspondent essentiellement en des métriques logicielles. Ainsi, un SQI est une variable dont la valeur peut être déterminée par une analyse directe des caractéristiques du produit ou du processus et dont la relation avec un ou plusieurs attributs est évidente. Cette variable doit être déterminée de la façon la plus objective possible. Ces variables peuvent être utilisées pour confirmer la contribution bénéfique d'une certaine propriété sur un attribut ou à l'inverse refléter un effet négatif sur un autre attribut. Il est intéressant de noter que cette architecture est différente de celles déjà existantes : « McCall's factor/criteria/metric » (McCal et al., 1977) et « Basili's goal/question/metrics » (Basili, 1992).

La Figure 1 .2 présente un aperçu graphique de l'architecture OPA.

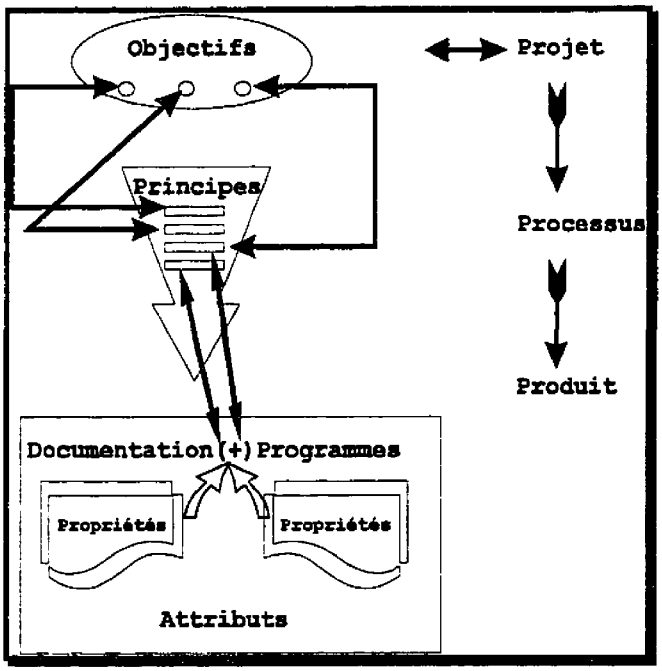


Figure 1 .2 : architecture de Arthur et al.

1 .4. Architecture de Dromey

Forêt

L'architecture proposée par Dromey (1995) est constituée de trois entités principales :

- un ensemble de composantes ;
- un ensemble d'attributs qui transportent certaines caractéristiques de la qualité ;
- un ensemble d'attributs de la qualité de haut niveau.

Il y a donc au plus six relations binaires entre ces entités. Or, puisque l'architecture est du type hiérarchique, seulement quatre de ces relations doivent être considérées. La Figure 1 .3 présente ces relations.

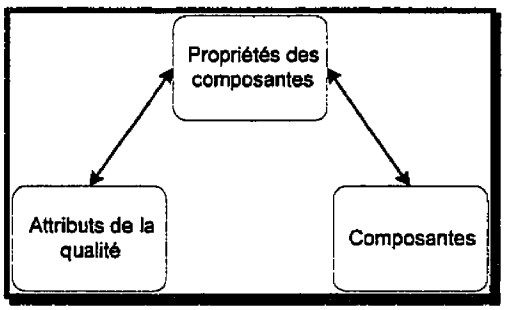


Figure 1 .3 : architecture de Dromey

À première vue, l'architecture de Dromey semble être une arborescence, mais dû au fait qu'elle est excessivement simpliste, il a fallu recourir à l'analyse de l'architecture sur laquelle est basé le modèle de qualité de Dromey et il s'est avéré que cette architecture était une forêt.

1 .5. Architecture de Kitchenham

N arborescences

Une autre approche alternative à l'architecture facteur-critère-métrique est proposée par Kitchenham (1987) et fait l'objet de la Figure 1 .4. Les flèches brisées de la figure ont pour objet d'indiquer que le facteur peut seulement être jugé contre les spécifications à la fin du développement, où il est possible de vérifier son accomplissement directement contre les spécifications grâce à des tests d'acceptance.

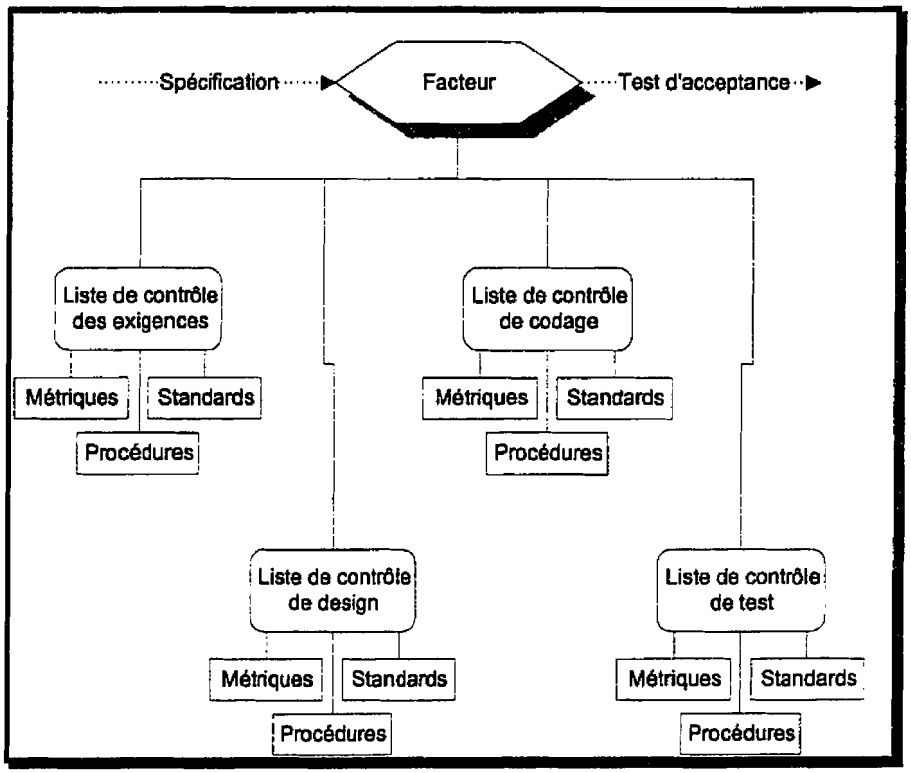


Figure 1 .4 : architecture de Kitchenham

La Figure 1 .4 indique que l'atteinte d'un facteur de qualité dépend du cycle de développement entier et qu'à chacune des étapes du cycle certaines listes de contrôle comprenant procédures,

standards et métriques devraient être utilisées pour contrôler le progrès envers le facteur de qualité.

La séparation des procédures, métriques et standards est une tentative pour identifier trois différentes caractéristiques en assurance qualité.

Les procédures sont les moyens par lesquels les ingénieurs de qualité essaient d'atteindre les exigences de qualité. Gilb (1986) mentionne qu'il est essentiel de comprendre quelles techniques logicielles permettent l'atteinte de certaines qualités. Présentement, ces relations sont des tentatives et elles ne sont pas prouvées de façon rigoureuse.

Les systèmes ou modules se conforment ou non à des standards. Donc, les standards sont des points vérifiables directement sur le produit ou produit intermédiaire. Les standards ne peuvent pas par conséquent être utilisés pour prédire la qualité durant le processus de développement. Cependant, ce n'est pas le cas pour des évaluations rétrospectives de la qualité où des standards vérificateurs peuvent être utilisés pour indiquer le pourcentage, le taux de conformité à certains standards comme une évaluation préliminaire de la qualité.

Les métriques sont des entités plus floues : elles peuvent être des mesures du produit ou du processus qui peuvent être utilisées comme indicateurs de problèmes ou prédire certains problèmes. Dans chacun des cas, une certaine norme est sous-entendue basée soit sur d'autres systèmes similaires ou sur d'autres modules dans le même système (par exemple : nous trouvons normalement 80% des erreurs par inspection et nous prévoyons que le nombre d'erreurs restant dans le système est X).

1 .6. Architecture de Kaposi et Kitchenham

Arborescence

Cette approche, présentée par Kaposi et Kitchenhem (1987), n'est pas limitée à la modélisation de la qualité logicielle ; elle englobe la notion de *système*. La définition formelle d'un système est : un système est un ensemble d'éléments liés formant une entité distincte à l'intérieur d'un environnement. Dans le présent contexte, un système peut inclure du matériel, du logiciel et quelques fois des éléments humains.

Une architecture adéquate pour représenter la qualité d'un système doit être suffisamment sophistiquée pour refléter la complexité liée à la notion de qualité et suffisamment claire pour établir une distinction entre les aspects objectifs (aspects absolus) et subjectifs (aspects qui captent les préférences individuelles) de la qualité. De plus, l'architecture doit être suffisamment générale pour les systèmes hybrides logiciel-matériel tout en étant facilement compréhensible et utilisable pour les pratiques logicielles quotidiennes.

Pour aider à atteindre ces objectifs, les concepts généraux de l'approche système ainsi que les définitions établies sur la qualité logicielle sont utilisés. Les auteurs reconnaissent que le premier essai à une approche systématique peut résulter en une architecture (modèle conceptuel) qui nécessitera des raffinements et des simplifications. Pour mettre l'architecture dans la pratique du génie logiciel, l'architecture générale devra être adaptée à des exigences spécifiques et supportée par un système de métriques adéquat.

D'une façon générale, l'architecture a été pensée pour réconcilier le concept subjectif de la qualité avec la nécessité industrielle de lier la qualité à des paramètres objectifs qui sont applicables dans un contrat.

1 .6.1. Aperçu de l'architecture

L'architecture impose une structure sur les propriétés de la qualité des produits et des processus. Elle sépare les aspects de la qualité qui peuvent être mis sous le régime de la

gestion de la qualité du design de ceux qui sont plus nébuleux et qui ne sont pas des préoccupations principales de l'ingénieur et ne pouvant être contrôlés directement.

En conséquence, la qualité d'un système est modélisée en définissant un nouveau concept soit le *profil de qualité*. Le profil de qualité d'un système est l'agrégat de trois classes distinctes de propriétés :

- propriétés transcendantales ;
- facteurs de qualité ;
- indices de mérites.

Chaque propriété à l'intérieur de chaque classe représente une caractéristique désirable du système du point de vue de la qualité.

Les auteurs partagent la même opinion que Pirsig (1974) et Garvin (1984) sur le fait qu'il y a des aspects de la qualité qui défient toutes définitions. De telles propriétés transcendantales ne peuvent être quantifiées ou analysées. Par conséquent, elles sont laissées de côté.

Les *facteurs de qualité* et *indices de mérites* sont des propriétés qui sont exprimées quantitativement. Elles sont définies de façon à ce que leur valeur augmente avec la majoration de la qualité. En général, les deux sont des propriétés de qualité composées dont la valeur ne peut normalement pas être obtenue par une évaluation, observation ou mesure directe. Au lieu de cela, elles sont définies en termes de paramètres du système plus simples et directement observables qui en eux-mêmes peuvent ne pas refléter la qualité.

Les *facteurs de qualité* sont définis en termes de paramètres du système neutres, objectifs, mesurables répétitivement et déterminables d'une façon non ambiguë. Les facteurs de qualité sont des fonctions définies au dessus de ces paramètres objectifs. Les fonctions peuvent refléter des vues subjectives et des préférences, des règles et codes de pratique bien définis ou des lois et théories scientifiques objectives. Les paramètres peuvent corréler avec les facteurs de qualité positivement ou négativement.

Les *indices de mérites* sont des fonctions définies en termes de quantités évaluées subjectivement qui sont appelées *classement* et qui servent de paramètres de base.

En conséquence, le *profil de qualité* d'un système est une structure à trois niveaux qui est présentée à la Figure 1 .5. Ce modèle schématique est indépendant du type, de l'implantation et d'autres propriétés du système mais peut facilement être adapté à des exigences spécifiques. Il représente ainsi une architecture pour caractériser la qualité d'un système. À partir de ce cadre de travail général, un profil de qualité différent peut être établi pour exprimer la vue des clients, des fournisseurs, des différents départements à l'intérieur de la compagnie du fournisseur ou par un évaluateur indépendant. À partir d'un certain profil de qualité spécialisé, comme celui représentant le point de vue du client, le modèle peut accommoder le goût personnel des individus, chacun pouvant choisir un certain produit pour différentes raisons.

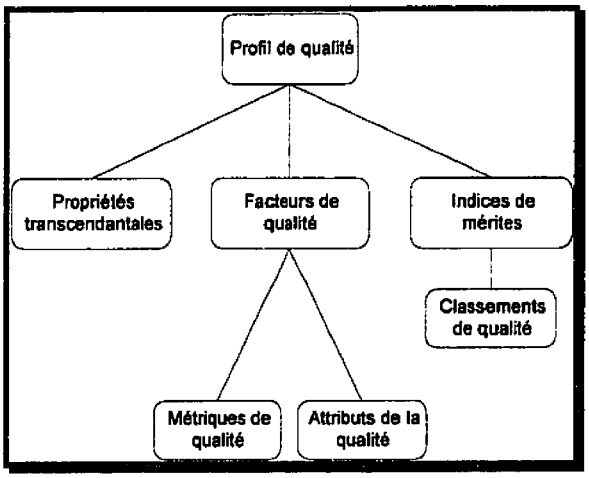


Figure 1 .5 : architecture de Kaposi et Kitchenham

1 .6.2. Analyse détaillée de l'architecture

On distingue deux types de paramètres objectifs selon lesquels on peut caractériser la qualité : *attributs* et *métriques*. Les *attributs de qualité* ne sont pas quantifiables ; ils indiquent seulement la présence ou l'absence d'une certaine propriété par l'entremise d'une variable booléenne. Les *métriques de qualité*, à l'opposé, quantifient le degré selon lequel une certaine propriété spécifique est présente dans le système.

Deux clients différents d'un même système décriront ce dernier en termes des mêmes métriques et attributs de qualité, mais ils peuvent définir les fonctions de leur facteur de qualité d'une façon différente, dépendant de leur point de vue individuel sur l'utilité du système. Ainsi, ils peuvent arriver à des conclusions différentes sur la qualité d'un même système.

Les *classements de qualité* sont des paramètres quantitatifs conçus pour capter le *jugement de valeur,* soit le degré selon lequel des individus ou un groupe d'individus approuvent certaines propriétés particulières du système. Par exemple, un individu peut décider d'assigner un classement à la facilité d'utilisation d'un certain logiciel. Ainsi, les utilisateurs peuvent alors indiquer leur vue subjective relativement à cette propriété en la classant sur une échelle de 0 à 10 où 0 signifie excessivement faible et 10 signifie une appréciation très positive.

Les *indices de mérites* sont des fonctions définies subjectivement dont la valeur quantifie un aspect spécifique de la qualité du système sous évaluation. Les arguments des fonctions d'*index de mérites* sont les classements de qualité du système.

On peut montrer l'utilisation des *classements de qualité* et des *indices de mérites* en prenant l'exemple de l'évaluation de projets finaux d'étudiants. Prenons pour acquis que chaque étudiant subventionné industriellement travaille sur un projet qui est un produit potentiel pour la compagnie qui le subventionne. Les projets sont évalués par des évaluateurs académiques et industriels. Six paramètres de classement sont utilisés : originalité, contenu analytique, contenu technologique, implantation, présentation et potentiel du produit proposé. Chaque classement se voit donner une valeur sur une échelle de 0 à 20. Les résultats obtenus sont utilisés pour calculer deux indices de mérites sur une échelle de 120 : la classification académique du projet et la probabilité que la compagnie poursuive son développement. Les deux indices de mérites peuvent consister en des sommes pondérées des classements, mais le poids relatif peut être différent dans les deux cas.

1 .6.3. Élaboration d'un modèle de qualité logicielle basé sur l'architecture de Kaposi et Kitchenham

La relation entre les *facteurs de qualité* et les *métriques/attributs de qualité* peut être modélisée comme une matrice de dépendance booléenne, où un '1' dans la matrice indique que le facteur de qualité de cette ligne dépend de la métrique ou de l'attribut de qualité de la colonne ; un '0' montre le contraire. Une matrice similaire de dépendance décrit les relations entre les *indices de mérites* et les *classements de qualité* dont ils dépendent. Il est à noter que nous n'avons pas considéré la matrice de dépendance booléenne dans le classement de l'architecture de Kaposi et Kitchenham ; ceci est justifié par le fait que nous considérons seulement l'architecture et non pas la façon d'implanter l'architecture. La considération de cette matrice aurait pour conséquence que l'architecture dégénérerait en un graphe simple.

La matrice de dépendance indique l'existence, mais non la nature, des relations entre la fonction et ses arguments. Dans le cas simple, chaque *facteur de qualité* peut être considéré comme dépendant linéairement de ces arguments. Dans ce cas, une matrice de qualité peut être définie dans laquelle un paramètre remplace chaque entrée non nulle de la matrice de dépendance. Un paramètre positif indique que le facteur de qualité de la ligne augmente avec l'attribut/métrique de la colonne ; un paramètre négatif indique l'inverse. La matrice de qualité qui modélise les relations entre les *indices de mérites* et les *classements de qualité* aurait un format similaire.

Dans la plupart des cas pratiques, certains *facteurs de qualité* (et *indices de mérites*) ne sont pas seulement des combinaisons linéaires pondérées de leurs arguments. Dans de tels cas, ils ne sont pas définis en terme d'une matrice de coefficients d'un ensemble d'équations linéaires simultanées. Au lieu de cela, ils sont donnés comme un ensemble de fonctions individuelles non linéaires dont les arguments sont des *attributs/métriques de qualité* (ou dans le cas des indices de mérites, des *classements de qualité*).

Les fonctions qui déterminent les *facteurs de qualité* et les *indices de mérites* peuvent être formulées algorithmiquement ou statistiquement. Dans les disciplines avancées, la construction de fonctions algorithmiques est dérivée d'une compréhension théorique des relations entre les paramètres directement mesurables et les propriétés dérivées du système.

Un exemple de fonction algorithmique est la formule *V* de Halstead (1977) qui est utilisée pour mesurer la complexité des programmes en terme de volume, soit

$$V=n \log_2 N$$

Selon l'architecture, l'expression pour *V* représente un *facteur de qualité* : une fonction définie à partir des paramètres *n* et *N* (qui sont des métriques objectives). Leur valeur est facilement calculable par une analyse statique des programmes. La fonction *V* elle-même est subjective : elle reflète la vision de Halstead qui n'est pas partagée par tous.

Comme nous l'avons mentionné précédemment, un *facteur de qualité* est considéré comme étant objectif si la fonction qui est utilisée pour déterminer sa valeur est dérivée de règles établies ou d'une théorie scientifique inductive ou déductive. Un *facteur de qualité* subjectif en est un qui n'est pas supporté par une théorie adéquate. Il peut refléter des opinions explicitement formulées ou des fonctions arbitraires sur des paramètres objectifs exprimant un goût individuel.

Les *indices de mérites* sont fonction des *classements de qualité*. Ils sont formulés algorithmiquement ou statistiquement de la même façon que pour les facteurs de qualité. Puisque les classements sont tous subjectifs, tous les indices de mérites sont subjectifs par nature.

Le *profil de qualité* global d'un système est une évaluation subjective de sa qualité. Ainsi, il est reconnu que les caractéristiques d'un système qui présentent un intérêt pour une personne peuvent ne présenter aucun intérêt ou être de priorité différente pour une autre personne.

Le *profil de qualité* du modèle proposé est consistant avec l'*approche basée sur le client* de Gilb (1984) (voir chapitre 1) et implique que les autres approches sont des éléments qui contribuent également à la vision de la qualité de l'usager. L'*approche transcendante* est directement liée aux propriétés transcendantales de l'architecture. L'*approche basée sur le produit* et l'*approche basée sur la valeur* comparent la valeur des facteurs de qualité et des indices de mérites pour classer la qualité des produits. L'*approche basée sur le processus* met de l'emphase sur les attributs/métriques et classements de qualité qui mesurent les caractéristiques du processus de production logicielle, incluant les caractéristiques statiques des produits intermédiaires.

ANNEXE II
MODÈLES DE QUALITÉ LOGICIELLE COURANTS

2.1. Introduction

Dans cette annexe, nous présentons les modèles de qualité logicielle les plus connus. Ces modèles appartiennent tous à la catégorie des modèles généraux. Conséquemment, ils sont applicables à n'importe quel type de systèmes logiciels. Les modèles sont présentés dans l'ordre de la classification que nous avons déjà élaboré. Il est à noter que nous avons conservé la terminologie originale pour les descriptions dans un souci de rapporter le plus fidèlement possible l'essence des modèles.

2 .2. Modèle COQUAMO de Kitchenham

Graphe simple

Le présent travail fait partie du projet REQUEST qui est subventionné par ESPRIT et est présenté par Kitchenham (1987). Le but du projet REQUEST est d'identifier des méthodes améliorées de fiabilité logicielle et d'évaluation de la qualité. Le projet est divisé en sous-projets dont un qui est concerné par la mesure, la modélisation et la prédiction de la qualité. Le but de ce sous-projet est de développer un modèle constructif de qualité (COQUAMO - COnstructive QUAlity MOdel) qui est dans un certain sens analogue au modèle de coût constructif de Boehm (COCOMO). Ainsi, il est prévu d'identifier un modèle prédictif de la qualité des produits basé sur les composantes dominantes de la qualité soient les attributs de l'organisation, personnel, produit, projet ou processus qui influencent la qualité des produits.

Un but plus précis du projet REQUEST est de lier les évaluations à des caractéristiques *mesurables* du logiciel et aussi à l'adhérence aux procédures. Pour formuler COQUAMO, les mesures de qualité suivantes sont nécessaires :

- mesures directes de la qualité logicielle ;
- mesures de l'accomplissement de la qualité durant le développement des produits ;
- mesures des composantes dominantes de la qualité en relation avec le produit, processus, organisation, personnel et projet et cela au début du projet.

Les mesures des composantes dominantes de la qualité sont nécessaires au début du projet pour fournir les paramètres d'entrée au modèle COQUAMO de base qui interagira avec les spécifications des exigences de qualité pour produire un modèle prédictif de qualité approprié pour des études de faisabilité.

Les mesures directes de qualité logicielle sont nécessaires pour permettre aux exigences de qualité d'être spécifiées au début du développement du produit et pour démontrer objectivement qu'elles sont atteintes ou non à la fin de la construction du produit. Les mesures de qualité durant le développement du produit sont importantes si le progrès de la production est pour être suivi et contrôlé avec succès.

Des problèmes surviennent entre les deux derniers types de mesure de qualité parce que les mesures prises durant le processus de production n'ont pas nécessairement de relations claires avec les mesures des attributs d'un produit fini. Par exemple, il n'y a pas de relation fonctionnelle évidente entre le nombre de fautes trouvées durant les inspections de design et la fiabilité du produit final en terme de probabilité d'exécution correcte du logiciel à une période précise. Il est toutefois possible d'identifier des relations statistiques.

Par conséquent, même si toutes les mesures reliées à la qualité peuvent être appelées métriques de qualité, il y a une différence importante entre les métriques liées à la qualité du produit final (appelées métriques de facteur de qualité) et les métriques liées à la production du logiciel (appelées indicateurs de qualité).

Même s'il apparaît que COQUAMO nécessite seulement des composantes dominantes de la qualité et des données d'indicateurs de qualité, les métriques de facteur de qualité sont aussi importantes pour les raisons suivantes :

- pour prédire quantitativement la qualité, il est nécessaire de formuler des mesures quantitatives ;
- pour établir des relations statistiques, il est nécessaire d'avoir des composantes dominantes de la qualité quantitative, des métriques de facteur de qualité et des indicateurs de qualité. Des relations statistiques sont nécessaires à la fois pour identifier des relations prédictives et pour valider l'influence des composantes dominantes de la qualité et d'indicateurs de qualité ;
- COQUAMO est destiné à être un modèle général et va certainement avoir besoin d'être recalibré pour un environnement particulier. Ceci impliquera l'établissement de relations statistiques particulières dans des environnements spécifiques.

Cette mise en situation sur le projet REQUEST est destinée à clarifier le point de vue des travailleurs sur la modélisation de la qualité qui sont concernés par les problèmes de la prédiction et de l'évaluation quantitative de la qualité. En particulier, REQUEST est concerné par les moyens par lesquels de telles prédictions et évaluations peuvent être utilisées pour assister la production de logiciels dans la mise en branle de projet et le processus de développement de produits tout au long du cycle de vie du logiciel. Le point de vue de l'utilisateur de logiciels est considéré en termes de ses entrées aux processus de spécification

des exigences de qualité et de ses entrées aux épreuves d'acceptation et de l'évaluation finale du produit.

Il est à noter que la formulation initiale de COQUAMO est prévue pour les projets de développement logiciel conventionnel utilisant des méthodes conventionnelles de développement. Cependant, étant donné que COQUAMO est concerné principalement avec la mesure et l'évaluation, il est à espérer que les effets de changer les méthodes de développement logiciel vont être incorporés naturellement. C'est-à-dire que l'utilisation de nouvelles techniques risque d'augmenter la faisabilité et de réduire le risque dans l'atteinte de certaines qualités désirables.

2 .2.1. L'approche REQUEST

L'approche prise par l'équipe du projet REQUEST vise le producteur logiciel et projette de fournir un modèle pour l'évaluation et l'atteinte de la qualité. Pour cela, les auteurs ont synthétisé le travail de Gilb (1986) et McCall et al. (1977) de la façon suivante :

- en modifiant le modèle simple de qualité proposé par McCall et al. pour y inclure les considérations de spécification de qualité, les concepts de cycle de vie et l'assurance qualité ;
- en rationalisant et classifiant les différents facteurs de qualité proposés par Boehm et al. (1978), McCall et al. (1977) et Bowen et al. (1984) ;
- en identifiant les facteurs de qualité qui sont généraux et directement mesurables dans le produit final et qui serviront de base pour un modèle général de qualité ;
- en identifiant les facteurs de qualité qui sont sous le contrôle du groupe de production et pouvant être contrôlés durant le développement de produits et en suggérant des critères sous-jacents et des métriques pour ces facteurs.

2 .2.2. Le développement de COQUAMO

Un modèle général de qualité peut seulement interagir avec des qualités qui sont applicables à la majorité des systèmes logiciels. Ainsi, les qualités d'un intérêt particulier pour COQUAMO sont la *fiabilité*, l'*utilisabilité* (acceptation par l'usager), la *maintenabilité*, la *réutilisabilité* et l'*extensibilité*. Cependant, un modèle constructif de qualité devrait permettre un suivi de la qualité durant le processus de production, ce qui implique que les qualités *testabilité*, *compréhensibilité* et la *justesse* présentent également un intérêt particulier.

Les auteurs stipulent que COQUAMO devrait être concerné par les deux types de facteurs de qualité, mais que les différents groupes de facteurs qualité seront traités d'une façon quelque peu différente.

Les auteurs suggèrent que les qualités telles la *fiabilité*, la *maintenabilité*, la *réutilisabilité*, l'*extensibilité* et l'*utilisabilité* (acceptation de l'usager) sont consistantes avec l'approche basée sur l'utilisateur de Garvin (1984). Ces qualités sont celles qui sont généralement spécifiées par l'utilisateur lors de la définition des exigences et qui peuvent seulement être démontrées lors de l'utilisation du système.

Présentement, il y a des techniques et des outils qui aident à atteindre ces qualités tels les méthodes formelles pour augmenter la *fiabilité* et des outils de traçage et de dévermillage pour la *maintenabilité*. Cependant, il n'est pas possible de mesurer le progrès vers ces qualités durant la production (même s'il peut être possible de faire des prédictions sur le comportement final du système en rapport avec ces qualités durant la phase de tests et les tests visant à démontrer des qualités devraient être planifiés).

COQUAMO fournira une assistance pour la spécification de ces qualités en utilisant les standards de spécification de qualité de Gilb (1986) et évaluera la faisabilité du *profil de qualité* requis en fonction des composantes principales de la qualité de haut niveau.

Les qualités telles la *testabilité*, la *compréhensibilité* et la *justesse* sont consistantes avec l'approche basée sur le producteur et aide le groupe de production à se conformer aux spécifications. De telles qualités sont également supportées par différents outils techniques et outils tels les standards de documentation et de codage pour la compréhensibilité, le traçage entre les spécifications et les plans de tests pour la testabilité et la démonstration de

programme pour la justesse. De plus, l'effet de ces pratiques peut être mesuré durant le processus de production logicielle.

COQUAMO fournira donc une assistance pour contrôler la qualité par l'analyse des métriques de qualité associées avec la *testabilité*, la *compréhensibilité* et la *justesse* qui peuvent être considérées comme des indicateurs de qualité. Il y a cependant certains problèmes associés avec les indicateurs de qualité qui font que l'élaboration d'un modèle basé seulement sur ces derniers n'offre pas tellement de potentiel :

- il y a rarement d'échelles objectives permettant d'interpréter les indicateurs de qualité ;
- le même indicateur de qualité (par exemple un taux d'erreur élevé pour un certain module) peut résulter de plusieurs causes différentes - par exemple, une politique très stricte de test pour un module, un module très mal codé, un mauvais enregistrement des données ou un module particulièrement complexe.

L'utilisation de COQUAMO pour contrôler la qualité logicielle nécessitera un processus pour interpréter les valeurs des mesures et un système pour interpréter de telles valeurs. En accord avec les procédures d'assurance qualité, les auteurs envisagent un système qui identifiera les tendances et les domaines potentiels de problèmes avec les causes correspondantes. Ainsi, les auteurs prévoient utiliser des techniques statistiques pour identifier les tendances telles l'augmentation ou la diminution du taux d'erreur, taux de codage, ... et les composantes anormales telles les modules avec un haut ou un bas taux d'erreur comparés aux autres modules.

2 .2.3. Représentation du modèle de Kitchenham

En analysant les discussions de la section précédente en conjoncture avec la classification des attributs externes de qualité ainsi que les liens entre ces attributs que Kitchenham a élaborés (voir chapitre 5), nous pouvons déduire un modèle de qualité qui refléterait la pensée de Kitchenham. Le modèle résultant est présenté à la Figure 2 .1.

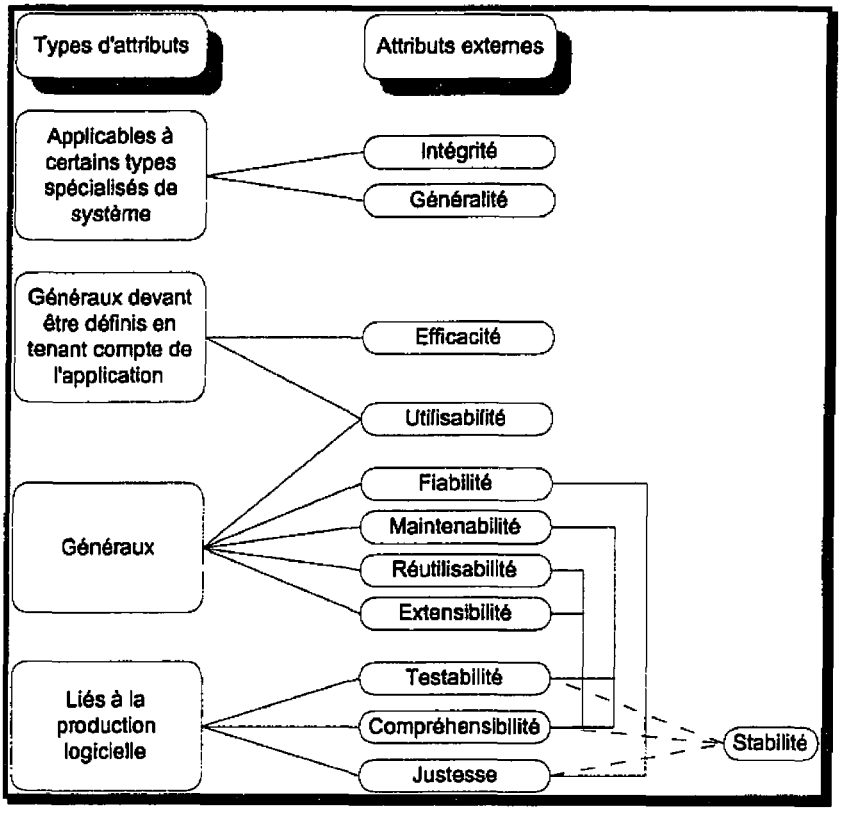


Figure 2 .1 : modèle de Kitchenham

2 .3. Modèle de Boehm, Brown, Kaspar, Lipow, MacLeod et Merrit

Graphe N-parti

Les objectifs initiaux de l'étude de Boehm et al. (1978) étaient d'identifier un ensemble de caractéristiques de la qualité logicielle et pour chacune d'elles de définir une ou plusieurs métriques telles que :

- ayant un programme arbitraire, la métrique fournit une mesure quantitative du degré selon lequel le programme possède la caractéristique associée, et
- la qualité logicielle globale peut être définie comme une certaine fonction des valeurs des métriques.

L'étude est divisée en quatre parties qui font l'objet des sous-sections suivantes.

2 .3.1. Phase initiale : quantification rapide

L'étude commence par formuler une liste de caractéristiques logicielles suivie de la formulation d'un grand nombre de quantités dérivées à partir du balayage d'un programme Fortran et qui semble avoir une certaine corrélation avec la qualité logicielle et, par la suite, vient la formulation d'une métrique globale pour la qualité logicielle comme une combinaison linéaire des quantités individuelles.

2 .3.2. Évaluation de la quantification rapide

Ensuite, une évaluation des résultats de la phase initiale fut réalisée. Plusieurs conclusions significatives ont émergé :

- pour presque toutes les formules quantitatives simples, il était facile de trouver des contre-exemples qui mettaient en doute leur crédibilité comme indicateurs de la qualité logicielle ;

- le champ du logiciel évolue encore trop rapidement pour établir des métriques dans certains domaines ;

- dans le développement et l'évaluation de produits logiciels, nous sommes généralement plus intéressés à savoir *où* et *comment* le produit est déficient que *combien souvent* le produit est déficient. Ainsi, les outils les plus utiles pour l'analyse logicielle sont généralement ceux qui révèlent les déficiences ou anomalies dans les programmes au lieu de ceux qui produisent seulement des nombres ;

- calculer et comprendre la valeur d'une métrique globale pour la qualité logicielle peut représenter plus de troubles que ce que les bénéfices représentent. Le problème majeur est que plusieurs des caractéristiques logicielles sont en conflit : augmenter l'*efficacité* est souvent obtenu au prix de la *portabilité*, de la *précision*, de la *compréhensibilité* et de la *maintenabilité*. Les clients trouvent qu'il est difficile de quantifier leurs préférences dans de telles situations conflictuelles. Un autre problème est que les métriques sont généralement des mesures incomplètes de leurs caractéristiques associées. Pour résumer ces considérations :

    - la qualité d'un produit logiciel varie avec les besoins et les priorités des clients ;
    - il n'y a pas, conséquemment, une métrique qui peut donner un classement universel de la qualité logicielle ;
    - au mieux, un client peut recevoir un classement utile en fournissant au système de classement un ensemble complet de listes de contrôle et de priorités ;
    - même encore, puisque les métriques ne sont pas exhaustives, le classement global résultant serait plus suggestif que concluant ou prescriptif ;
    - par conséquent, la meilleure utilisation des métriques à ce point est comme indicateurs d'anomalies qui peuvent être utilisées comme guide pour le développement et la maintenance logiciels.

- la plupart des ensembles de caractéristiques logiciels sont définis de façon approximative et présentent trop de chevauchement pour être d'une certaine utilité

pratique. Ce qui a amené les auteurs à fournir un effort soutenu en vue de définir précisément l'ensemble des caractéristiques et de leurs interrelations entre elles et aussi de raffiner l'ensemble des métriques de qualité.

2 .3.3. Phase finale : caractéristiques hiérarchiques et métriques détectrices d'anomalies

En se basant sur les conclusions de la dernière section, l'étude suit l'approche qui fut précédemment décrite dans le chapitre 3 pour développer un ensemble de caractéristiques hiérarchiques et un ensemble de métriques détectrices d'anomalies. Ainsi, selon cette approche, un ensemble original de caractéristiques candidates importantes pour le logiciel fut défini. Par la suite, un ensemble de métriques candidates pour évaluer le degré selon lequel le logiciel possède les caractéristiques de qualité fut développé.

L'ensemble original des caractéristiques est le suivant :

1. compréhensibilité ;
2. complétude ;
3. concision ;
4. portabilité ;
5. consistance ;
6. maintenabilité ;
7. testabilité ;
8. utilisabilité ;
9. fiabilité ;
10. structure ;
11. efficacité.

En évaluant les métriques formulées pour l'ensemble des caractéristiques ci-haut, un haut degré de chevauchement fut trouvé. Par exemple, 11 des 14 métriques d'*utilisabilité*, 5 des 8 métriques de *fiabilité* et 7 des 13 métriques de *structure* avaient déjà été associées avec d'autres caractéristiques de l'ensemble. Également, l'ensemble n'était pas complet. Par exemple, à part les 23 métriques de *compréhensibilité*, 25 autres métriques furent trouvées corrélant avec la *compréhensibilité* mais étaient disponibles seulement sous une autre

caractéristique. Dans le processus d'évaluation, un ensemble plus primitif de caractéristiques fut déterminé avec moins de chevauchement. Un modèle fut également trouvé sur les relations entre l'ensemble original de caractéristiques, l'ensemble révisé et où se situe l'utilisation de l'évaluation de la qualité logicielle. Ainsi, ayant une partie d'un logiciel opérant dans un environnement donné, il y a principalement trois choses que quelqu'un peut vouloir faire avec le logiciel. Elles sont :

1. *l'utiliser comme il est*
    - comprendre ce que le programme peut faire ;
    - obtenir des résultats fiables :
        - déterminer et corriger les sources de non fiabilité (mène à #2) ;
    - utiliser les ressources efficacement :
        - ressources humaines
        - ressources matérielles.
2. *le maintenir*
    - comprendre la structure du programme ;
    - développer et incorporer des modifications ;
    - tester pour s'assurer de la fiabilité ;
    - utiliser les ressources efficacement :
        - ressources humaines
        - ressources matérielles.
3. *l'utiliser dans un autre environnement*
    - évaluer l'utilisabilité (mène à #1) ;
    - évaluer la maintenabilité (mène à #2) ;
    - évaluer la portabilité ;
    - convertir le programme pour qu'il opère dans un autre environnement.

Idéalement, les auteurs désirent à partir des caractéristiques se rendre à un ensemble d'attributs numériques qui identifierait le degré selon lequel le logiciel possède les caractéristiques. Cependant, en essayant d'y arriver, les auteurs ont dû formuler une hiérarchie de caractéristiques de plus en plus primitives où le niveau le plus haut correspond plus naturellement aux besoins d'utilisation et où les niveaux inférieurs sont plus faciles à mesurer. Le modèle résultant est présenté à la Figure 2 .2.

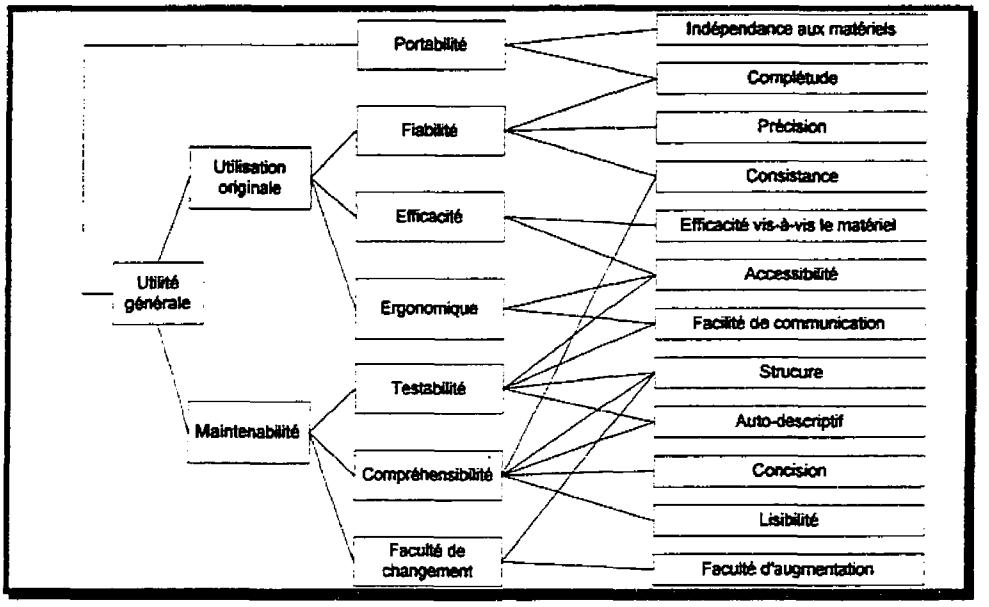


Figure 2 .2 : modèle Boehm et al.

Toutes les caractéristiques originales demeurent inchangées dans le modèle de la Figure 2 .2. La seule exception est que l'*utilisabilité* est séparée en l'*utilisation originale* qui réfère à l'usage du produit logiciel seulement dans sa forme courante et avec les systèmes informatiques en place et l'*utilité générale* qui réfère à l'usage et aux modifications du produit logiciel pour servir à la fois les systèmes informatiques en place et les autres systèmes informatiques. Le Tableau 2 .1 présente les définitions de chacune des caractéristiques du modèle.

L'idée générale du modèle est la suivante : le niveau le plus haut de la structure reflète l'usage d'un produit logiciel. Ainsi, la *portabilité*, l'*utilisation originale* et la *maintenabilité* sont des conditions nécessaires et suffisantes pour l'*utilité générale*. L'*utilisation originale* requiert qu'un programme soit *fiable*, *efficace* et *ergonomique* mais ne requiert pas que l'utilisateur teste le programme, comprenne son fonctionnement interne, le modifie ou essaie de l'utiliser dans un autre environnement. La *maintenabilité* requiert que l'utilisateur soit capable de *comprendre*, *modifier* et *tester* le programme, mais ne dépend pas de la *fiabilité*, l'*efficacité* ou la *portabilité* courante du programme. La structure du plus bas niveau du modèle fournit un ensemble de caractéristiques primitives qui sont également fortement différenciées les unes

par rapport aux autres et qui forment un ensemble de conditions nécessaires et suffisantes pour les caractéristiques du niveau intermédiaire.

Les caractéristiques primitives définies fournissent une fondation pour définir des métriques quantitatives qui peuvent ensuite être utilisées pour mesurer la présence à la fois des primitives et des caractéristiques de haut niveau en termes qui peuvent aider l'évaluation de l'utilité des produits logiciels (à un haut niveau) et en prescrivant des directions pour l'amélioration (au niveau des primitives). Il y a un total de 151 métriques candidates qui sont présentées avec leur caractéristique associée dans le Tableau 2 .2. Toutes les métriques sont sous forme de questions pouvant être répondues par oui ou non. Ces métriques ont été développées pour des programmes sources écrits en Fortran qui utilisaient encore des cartes perforées, ce qui les rend parfois non applicables aux nouveaux langages de programmation.

Tableau 2 .1 : définitions des caractéristiques du modèle de Boehm et al.

Niveau	Caractéristique	Définitions
1	**Utilité générale + utilisation originale = utilisabilité**	Un produit logiciel possède la caractéristique *utilisabilité* selon qu'il est commode et que l'utilisation de toutes ses fonctionnalités est possible.
	Maintenabilité	Un produit logiciel possède la caractéristique *maintenabilité* selon qu'il facilite l'évolution pour satisfaire les nouvelles exigences.
2	**Portabilité**	Un produit logiciel possède la caractéristique *portabilité* selon qu'il peut opérer facilement et efficacement sur des configurations informatiques autres que celle courante.
	Fiabilité	Un produit logiciel possède la caractéristique *fiabilité* selon qu'il est possible de s'attendre à ce qu'il exécute ses fonctions promises d'une façon satisfaisante.
	Efficacité	Un produit logiciel possède la caractéristique *efficacité* selon qu'il remplit ses objectifs sans gaspiller de ressources.
	Ergonomique	Un produit logiciel possède la caractéristique *ergonomique* selon qu'il remplit ses objectifs sans gaspiller le temps, l'énergie ou le moral des utilisateurs.
	Testabilité	Un produit logiciel possède la caractéristique *testabilité* selon qu'il facilite l'établissement de critères d'acceptance et supporte l'évaluation de ses performances.

	Compréhensibilité	Un produit logiciel possède la caractéristique *compréhensibilité* selon la clarté du produit pour l'évaluateur.
	Faculté de modification	Un produit logiciel possède la caractéristique *faculté de modification* selon qu'il facilité l'incorporation de changements une fois que la nature des changements désirés est déterminée.
3	**Indépendance aux matériels**	Un produit logiciel possède la caractéristique *indépendance aux matériels* selon qu'il peut être exécuté sur des configurations informatiques autres que celle courante.
	Complétude	Un produit logiciel possède la caractéristique *complétude* selon que toutes ses parties sont présentes et que chaque partie est complètement développée.
	Précision	Un produit logiciel possède la caractéristique *précision* selon que ses sorties sont suffisamment précises pour qu'elles satisfassent leurs utilisations promises.
	Consistance	Un produit logiciel possède la caractéristique *consistance interne* selon qu'il contient une notation, une terminologie et une symbologie uniforme. Le produit possède la caractéristique *consistance externe* selon que le contenu peut être relié aux exigences.
	Accessibilité	Un produit logiciel possède la caractéristique *accessibilité* selon qu'il facilite l'utilisation sélective de ses composantes.
	Faculté de communication	Un produit logiciel possède la caractéristique *faculté de communication* selon qu'il facilite la spécification des entrées et fournit des sorties dont la forme et le contenu sont facilement compréhensibles et utiles.
	Structure	Un produit logiciel possède la caractéristique *structure* selon qu'il possède un modèle défini d'organisation de ses parties interdépendantes.
	Auto-descriptif	Un produit logiciel possède la caractéristique *auto-descriptif* selon qu'il contient assez d'informations pour qu'un lecteur puisse déterminer ses objectifs, ses hypothèses, ses contraintes, ses entrées, ses sorties, ses composantes et son statut.
	Concision	Un produit logiciel possède la caractéristique *concision* selon qu'il n'y a pas d'information excessive présente.
	Lisibilité	Un produit logiciel possède la caractéristique *lisibilité* selon que ses fonctions et que ses énoncés sont facilement discernables en lisant le code.

	Faculté d'augmentation	Un produit logiciel possède la caractéristique *faculté d'augmentation* selon qu'il accommode facilement les expansions au niveau structure des données ou au niveau fonctionnel.

Tableau 2 .2 : métriques de qualité du modèle de Boehm et al.

Compréhensibilité	1.1. Est-ce que les noms des variables décrivent les propriétés physiques ou fonctionnelles représentées ? 1.2. Est-ce que le code mort est adéquatement expliqué ? 1.3. Est-ce que les entrées, sorties et hypothèses des modules du programme sont clairement déclarées par l'entremise d'une liste de paramètres ? 1.4. Est-ce que chaque module du programme contient un bloc d'en-tête contenant : • nom du programme ; • date effective ; • exigences précises ; • objectifs ; • limitations et restrictions ; • historique des modifications ; • entrées et sorties ; • méthode ; • hypothèses ; • procédures de récupération pour toutes les erreurs prévisibles qui peuvent exister. 1.5. Est-ce que les fonctions reconnaissables contiennent une description adéquate pour que l'objectif de chacune soit clair ? 1.6. Est-ce que les points de décisions et les alternatives de branchements subséquentes sont décrits adéquatement ? 1.7. Est-ce qu'il y a des descriptions permettant la corrélation des noms des variables avec les propriétés physiques ou les entités qu'elles représentent ? 1.8. Est-ce que toutes les conditions de terminaison anormale sont décrites ? 1.9. Est-ce que des procédures de récupération en cas d'erreurs sont incluses et décrites ? 1.10. Est-ce que le programme contient des modules séparés qui ont été référencés à plusieurs endroits au lieu d'avoir des blocs de code redondants ? 1.11. Est-ce qu'il y a des listes de renvois des noms de variables ?

	Est-ce qu'il y a une carte des sous-routines appelées et appelantes ? 1.12. Est-ce que des indentations, lignes blanches, lignes ou boites de X ou S sont utilisées pour séparer les parties de code ? 1.13. Est-ce que les déviations du flot logique vers l'avant sont adéquatement décrites ? 1.14. Est-ce que l'ordonnancement suivant est suivi : bloc d'en-tête, énoncés de spécifications (de déclarations) et code exécutable ? 1.15. Est-ce qu'il y a un transfert à tous les énoncés étiquetés et est-ce que ces transferts sont facilement localisables ? 1.16. Est-ce que tous les éléments d'un tableau sont fonctionnellement reliés ? 1.17. Est-ce que des parenthèses ou d'autres techniques ont été utilisées pour éliminer l'ambiguïté dans l'ordre ou le mode d'évaluation des expressions arithmétiques ? 1.18. Est-ce qu'il y a au plus une assignation de paramètres par ligne de code ? 1.19. Est-ce qu'il y a au plus un énoncé exécutable par ligne de code ? 1.20. Est-ce que des indices sont utilisés dans toutes les utilisations de tableaux dimensionnés ? 1.21. Est-ce que les commentaires descriptifs ont un "fog index" bas ? [G62] 1.22. Est-ce que les programmes ou éléments de programmes qui utilisent ce programme ou élément sont identifiés ? 1.23. Est-ce que les programmes ou éléments de programmes qui sont utilisés par ce programme ou élément sont identifiés ?
Complétude	2.1. Est-ce que le programme contient (1) du code protégeant les opérations potentiellement non définies (i.e., division par zéro) (2) des commentaires adéquats pour définir les conditions sous lesquelles des opérations non définies sont possibles ? 2.2. Est-ce que l'étendue des boucles est testée avant qu'elles ne soient utilisées ? ("Are loop and multiple transfer index parameters range tested before they are used ?") 2.3. Est-ce que le programme contient tous les sous-programmes référencés non disponibles dans la librairie usuelle de systèmes ? 2.4. Est-ce qu'il n'y a pas de sous-programmes "dummy" références ? 2.5. Est-ce que l'étendue des indices est testée avant qu'ils soient utilisés ? 2.6. Est-ce qu'il y a des options disponibles à l'utilisateur de sorte

	que des calculs ou des caractéristiques sélectionnés peuvent être demandés ? 2.7. Est-ce que le programme a la capacité d'exhiber des messages d'erreurs clairs et utiles si des erreurs surviennent ? 2.8. Est-ce que toutes les entrées sont utilisées à l'intérieur du programme ; dans le cas contraire est-ce que leur présence est expliquée par des commentaires ?
Concision	3.1. Est-ce que tout le code est atteignable ? 3.2. Est-ce que le programme est exempt de sous-programmes "dummy" ? 3.3. Est-ce que le programme est exempt de tableaux multi-dimensionnels qui pourraient être remplacés par un plus grand nombre de tableaux mais dont chacun aurait moins de dimensions ? 3.4. Est-ce que les expressions composées qui sont répétées dans les énoncés arithmétiques sont évitées ? 3.5. Est-ce que les formats d'entrées/sorties spéciaux ou non nécessaires sont évités ? Si non, est-ce qu'un format déjà existant peut être utilisé en remplacement sans perte d'informations ? 3.6. Est-ce que les commentaires du programme ont un "fog index" bas ? 3.7. Est-ce que le code redondant est évité par l'exécution répétée d'une section de code ou d'un module ? 3.8. Est-ce que le programme supporte l'omission d'entrées, de calculs ou de sorties inutiles en mode de fonctionnement optionnel ? 3.9. Est-ce que les boucles sont exemptes de calculs qui n'affectent pas l'index de la boucle ? 3.10. Est-ce que les étiquettes d'énoncés sont assignées seulement à ceux qui en ont besoin ? 3.11. Est-ce que les décisions binaires sont utilisées lorsque les expressions de branchement peuvent être évaluées par vrai ou faux ?
Portabilité	4.1. Est-ce que tous les indices de paramètres apparaissent dans un énoncé de spécification qui identifie le paramètre comme un tableau dimensionné ? ("Do all subscripted parameters appear in a specification statement which identifies the parameters as a dimensioned array ?") 4.2. Est-ce que les noms de variables correspondant à des jetons du langage sont évités ? 4.3. Est-ce que les calculs sont indépendants de la grosseur des

	mots de la machine pour atteindre la précision requise ? 4.4. Est-ce que le programme nécessite une initialisation de la mémoire à être utilisée ? 4.5. Non applicable 4.6. Est-ce que le langage de programmation est supporté par plus d'un manufacturier d'ordinateurs ? 4.7. Est-ce que le programme est écrit à partir d'un ensemble standard de constructions disponibles dans le langage de programmation utilisé (préférablement un standard national tel ANSI ou IBM) ? 4.8. Est-ce que le programme dépend de librairies de systèmes ou de routines uniques à une installation particulière ? 4.9. Est-ce que les énoncés dépendant de la machine ont été notés et commentés ? 4.10. Est-ce que le programme est structuré pour permettre l'opération en phase sur une machine plus petite ? ("Is the program structured in a fashion which allows phased operation on a smaller computer ?") 4.11. Est-ce que les dépendances sur la représentation interne des bits des caractères alphanumériques ou spéciaux ont été évitées ou sinon ces dépendances ont-elles été notées et commentées ? 4.12. Si le codage selon la machine est utilisé (i.e., pour l'efficacité), est-ce qu'il y a une version du programme à jour dans un langage de plus haut niveau qui est disponible et maintenue ? 4.13. Est-ce que l'utilisation de caractéristiques spéciales de la machine est isolée ?
Consistance	5.1. Est-ce qu'un nom de variable est utilisé pour représenter des entités physiques différentes dans le programme ? Est-ce que les entités physiques différentes dans le programme sont représentées par des noms différents plutôt que par une seul nom de variables ? 5.2. Est-ce que la même entité physique est représentée par un seul nom au lieu de deux ou plusieurs noms de variables ? 5.3. Est-ce que toutes les spécifications (de déclarations) des ensembles de variables globales sont identiques (i.e., étiquetées COMMON) ? 5.4. Est-ce que le programme contient une seule représentation pour les constantes physiques ou mathématiques ? 5.5. Est-ce que les expressions arithmétiques similaires sont construites d'une façon similaire ? 5.6. Est-ce que la même étiquette (descripteur mnémonique) est toujours utilisée pour identifier la même sortie du programme ? 5.7. Est-ce que deux ou plusieurs étiquettes (descripteurs

	mnémoniques) sont utilisées pour étiqueter des sorties de programme différentes au lieu de la même étiquette ? 5.8. Est-ce qu'une méthode d'indentation (dans le listage source) consistante est utilisée ? 5.9. Est-ce que le type d'une variable est consistant avec toutes ses utilisations ? 5.10. Est-ce que les tolérances sur les itérations, intégrations, etc. sont consistantes avec le nombre de chiffres significatifs dans les entrées et sorties ? 5.11. Est-ce que le style et le format des commentaires sont consistants dans tous les modules du programme ? 5.12. Est-ce que tous les éléments d'un tableau sont fonctionnellement reliés ? 5.13. Est-ce que l'usage des fonctions est consistant avec leur type défini ?
Maintenabilité	6.1. Est-ce que le programme évite d'utiliser des expressions arithmétiques avec des constantes littérales qui sont sujettes à changer ? 6.2. Est-ce que certaines règles ont été employées dans l'étiquetage de blocs de code fonctionnellement distincts ? 6.3. Est-ce que l'étiquetage des énoncés est dans l'ordre alphanumérique ascendant ? 6.4. Est-ce qu'une certaine partie de mémoire a été réservée pour supporter les extensions en vue de fournir des capacités additionnelles ? 6.5. Est-ce que la recherche est facile à travers le listage pour trouver l'étiquette à laquelle un énoncé de branchement peut transférer le contrôle ? 6.6. Est-ce que la recherche à travers le listage est facile pour trouver où une variable utilisée dans une expression est assignée ? 6.7. Est-ce que la recherche est facile à travers le listage pour trouver tous les branchements qui peuvent transférer le contrôle à un certain énoncé étiqueté ? 6.8. Est-ce que le programme est subdivisé en modules selon des fonctions facilement reconnaissables ? 6.9. Est-ce que les listes de renvois des noms de variables générées par la machine sont fournies comme de la documentation standard ? ("Are machine-generated cross-reference listings of variables names with routine supplied as standard documentation ?") 6.10. Est-ce qu'il est possible de modifier les constantes du programme qui sont sujettes à des changements ?

Critère d'évaluation	Métrique
	6.11. Est-ce qu'il y a de l'information pour supporter l'évaluation de l'impact d'un changment dans d'autres portions du programme ? 6.12. Est-ce qu'il y a de l'information pour supporter l'identification du code du programme qui doit être modifié pour effectuer le changement requis ? 6.13. Est-ce qu'il y a au plus une assignation de paramètres par ligne de code ? 6.14. Est-ce qu'il y a au plus un énoncé exécutable par ligne de code ? 6.15. Est-ce que chaque bloc COMMON (énoncé de déclaration) contient seulement des données reliées ? 6.16. Est-ce que le programme a été structuré en des modules fonctionnellement indépendants ? 6.17. Est-ce que les endroits où il y a dépendance des modules sont clairement spécifiés par des commentaires ? ("Where there is module dependence, is it clearly specified by commentary, program documentation, or inherent program structure ?") 6.18. Est-ce que le format et l'ordonnancement des énoncés de spécifications (de déclarations) sont consistants dans tous les modules du programme ?
Testabilité	7.1. Est-ce que le programme contient des dispositions pour exhiber des résultats intermédiaires sur demande ? 7.2. Est-ce que toutes les sorties du programme sont identifiées avec des étiquettes descriptives ? 7.3. Est-ce que le programme contient des dispositions pour exhiber l'identification des cas de tests et/ou la description des résultats des tests ? 7.4. Est-ce que les sorties clés du programme sont identifiées et exhibées d'une façon permettant l'évaluation sommaire des résultats des tests ? 7.5. Est-ce que le programme fournit une pagination logique des résultats imprimés pour supporter l'examen visuel? 7.6. Non applicable. 7.7. Est-ce que le programme facilite l'exécution de cas de tests multiples sans avoir à spécifier d'une façon redondante les valeurs d'entrées qui ne changent pas ? 7.8. Est-ce que le programme a la capacité de tracer et exhiber le flot logique de contrôle ? 7.9. Est-ce que le programme peut être réexécuter à partir de points de reprise ? 7.10. Est-ce que le programme peut exhiber les étiquettes de toutes les entrées sur demande ? (Does the program provide for

	labeled display of all inputs upon request?) 7.11. Est-ce que tous les sous-programmes contiennent au plus un point de sortie ? 7.12. Est-ce que le programme peut exhiber des messages de diagnostics si des erreurs surviennent ? 7.13. Est-ce que le contrôle du séquencement des sous-programmes est spécifiable explicitement à partir des entrées ? 7.14. Est-ce que les fonctions des modules et leurs entrées/sorties sont définies adéquatement pour permettre le test des modules ? 7.15. Est-ce qu'il y a des commentaires pour supporter la sélection de valeurs d'entrées spécifiques permettant l'exécution de tests spécialisés ? 7.16. Est-ce que toutes les entrées/sorties des sous-programmes sont identifiées et décrites ? 7.17. Est-ce que le programme supporte l'omission d'entrées, de calculs ou de sorties inutiles en mode de fonctionnement optionnels ?
Utilisabilité	8.1. Est-ce que le programme fournit une pagination logique des résultats imprimés pour supporter l'examen visuel? 8.2. Est-ce que le programme peut être utilisé répétitivement sans avoir à spécifier d'une façon redondante les valeurs d'entrées couramment utilisées ? 8.3. Est-ce que le programme peut réinitialiser les variables avant une utilisation subséquente ? 8.4. Non applicable 8.5. Est-ce que le programme reconnaît la fin des entrées sans que l'usager ait à compter et spécifier le nombre d'entrées ? 8.6. Est-ce que le programme peut être réexécuter à partir de points de reprise ? 8.7. Est-ce que le programme permet l'utilisation d'entrées de différentes formes ? 8.8. Est-ce qu'il y a une variété de sorties disponible à l'usager ? 8.9. Est-ce que les formats des sorties sont suffisamment descriptifs pour être interprétés rapidement sans avoir à référer à d'autres documents ? 8.10. Est-ce que le programme fournit des diagnostics d'erreurs compréhensibles ? 8.11. Est-ce que le contrôle du programme est explicitement spécifiable par les paramètres d'entrées ? 8.12. Est-ce que le programme possède un utilitaire de récupération en cas d'erreurs non fatales ? 8.13. Est-ce que le programme a la capacité d'assigner des valeurs

	par défaut aux paramètres non spécifiés ? 8.14. Est-ce que le programme supporte l'omission d'entrées, de calculs ou de sorties inutiles en des modes de fonctionnement optionnels ?
Fiabilité	9.1. Est-ce que le programme est exempt d'erreurs évidentes ? 9.2. Est-ce que les résultats des tests sont acceptables ? 9.3. Est-ce que des procédures de récupération en vue d'erreurs sont incluses ? 9.4. Est-ce qu'il y a évidence que tous ou presque tous les chemins logiques ont été utilisés avec succès ? 9.5. Est-ce que les méthodes numériques utilisées par le programme sont suffisamment précises pour une application spécifique ? 9.6. Est-ce que chaque variable est utilisée dans le même mode tout au long du programme ? 9.7. Est-ce que les données d'entrées sont vérifiées pour les erreurs ? 9.8. Est-ce que l'étendue des boucles est testée avant leur utilisation ? ("Are loop and multiple transfer index parameters range-tested before use ?") 9.9. Est-ce que l'étendue des indices est testée avant leur utilisation ? 9.10. Est-ce qu'il y a une protection contres les opérations arithmétiques potentiellement non définies ?
Structure	10.1. Est-ce que certaines règles ont été employées dans l'étiquetage de blocs de code fonctionnellement distincts ? 10.2. Est-ce que les énoncés sont étiquetés dans l'ordre alphanumérique ascendant ? 10.3. Est-ce que les énoncés de spécifications (de déclarations) et les énoncés de format sont groupés ensemble d'une façon consistante à une location physique similaire dans tous les sous-programmes (i.e., toujours au début, etc.) ? 10.4. Est-ce que le contenu d'un bloc commun est fonctionnellement relié ? 10.5. Est-ce que les paires de blocs étiquetés sont visiblement séparées par des commentaires d'espacement? 10.6. Est-ce que les types d'énoncés distincts de spécifications (de déclarations) sont organisés dans un format consistant et ordonné ? 10.7. Est-ce que les énoncés de boucles DO imbriqués et les autres éléments structurels sont indentés pour indiquer les niveaux de hiérarchie ?

	10.8. Est-ce que le flot du programme est toujours vers l'avant avec les exceptions dûment commentées ? 10.9. Est-ce que tous les sous-programmes et fonctions ont un seul point d'entrée ? 10.10. Est-ce que le programme est exempt de blocs de code inutilisés ? 10.11. Est-ce que la structure de recouvrement est consistante avec le séquencement des sous-programmes ? 10.12. Est-ce que les modules sont limités en grosseur ? 10.13. Est-ce que des règles de transfert de contrôle entre les modules ont été établies et suivies ?
Efficacité	11.1. Est-ce que l'évaluation redondante d'expressions composées est évitée ? 11.2. Est-ce que tous les calculs non dépendants d'une boucle résident à l'extérieur des boucles ? 11.3. Est-ce que le programme peut être réexécuté à partir de points de reprise ? 11.4. Est-ce que le programme permet la modification de l'utilisation des ressources ; i.e., par l'entremise de tableaux de dimension variable ? 11.5. Est-ce que le programme permet le calcul selon des précisions variables ? 11.6. Est-ce que les sous-programmes qui sont fréquemment utilisés sont optimisés pour la vitesse d'exécution ? 11.7. Est-ce que les paramètres des sous-routines sont passés par des variables globales plutôt que par une séquence d'appels ? 11.8. Non applicable 11.9. Non applicable 11.10. Est-ce que la notation entière est utilisée pour les exposants ? 11.11. Est-ce que les blocs souvent utilisés sont combinés en des sous-routines ?

À ce point-ci, les auteurs ont développé :

- plusieurs questions pour juger la qualité d'un produit logiciel ;
- une caractérisation des qualités auxquelles les questions sont liées ainsi que les relations de ces qualités à des considérations d'utilité à un niveau plus élevé ;
- un ensemble de critères pour évaluer la pertinence des métriques.

Ce qui est requis par la suite est le développement d'algorithmes détaillés pour répondre aux questions et une relation entre les résultats numériques des algorithmes et le processus de

décision sur le développement et la procuration de logiciels en se basant sur la qualité logicielle. Même si ces développements s'avèrent assez directs, les auteurs ont présenté seulement un algorithme détaillé et une discussion de son utilisation selon une métrique, soit les commentaires d'en-tête (voir Boehm et al. (1978)).

2.3.4. Résultats majeurs

Les résultats majeurs de cette étude furent les suivants :

- une évaluation des limites des mesures purement quantitatives de la qualité logicielle ;
- une hiérarchie de caractéristiques logicielles, en d'autres termes le modèle de qualité ;
- une liste de métriques détectrices d'anomalies qui furent minutieusement classifiées selon les caractéristiques de la qualité logicielle et évaluées selon certains critères ;
- un algorithme et un guide pour utiliser les métriques dans l'évaluation des en-têtes des programmes ;
- une discussion et une analyse des coûts-bénéfices de l'utilisation des métriques durant le processus de développement logiciel (voir Annexe IV).

Cependant, selon Boehm et al.(1978), la contribution la plus importante de leur étude a été de fournir pour la première fois un modèle bien défini pour évaluer le sujet complexe de la qualité logicielle. Toujours selon eux, le modèle n'est certainement pas complet, mais il a été mené à un point suffisant pour supporter l'évaluation de l'efficacité (en terme de coût) des outils d'analyse de code étudiés dans cette étude.

2 .4. Modèle de Arthur, Nance et Balci

Graphe N-parti

Le but du modèle de Arthur et al. (1993), bien qu'étant très générique, est d'établir une approche pour contrôler le développement logiciel. Les parties les plus importantes de l'approche sont les suivantes :

- reconnaître que les processus de développement et de maintenance sont inévitablement liés ;
- formulation d'un modèle qui capture succinctement les dépendances entre et à l'intérieur des phases du cycle de développement ;
- design d'un système (semi) automatique de collecte des métriques ;
- définition d'une méthode de contrôle de processus qui supporte les décisions techniques et de gestion.

De plus, le modèle est conçu pour manipuler à la fois les indicateurs de produits et ceux de processus. Le modèle est basé sur l'architecture OPA (« Objectives/Principles/Attributes ») que nous avons étudiée à l'annexe A.

Un ensemble de relations entre les objectifs, principes et attributs est présenté à la Figure 2 .3. Comme on peut le constater, les objectifs sont liés à plus d'un principe et chaque principe est lié à plus d'un attribut. En fait, il y a 33 liens entre les objectifs et les principes et il y a 24 liens entre les principes et les attributs. Dans ce schéma, les SQI ("Software Quality Indicators") ne sont pas présentés, mais ils le sont dans un autre article de Arthur et Nance (1987). On sait cependant qu'il y a 87 SQI.

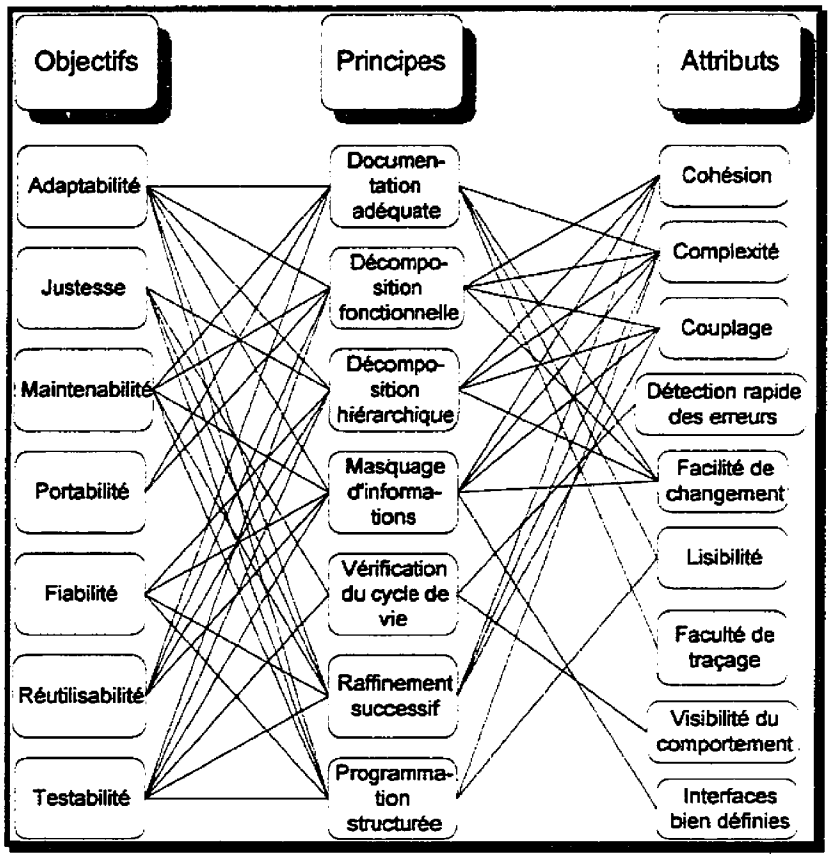


Figure 2 .3 : modèle de Arthur et al.

2.5. Modèle de Dromey

Forêt

Le modèle de qualité proposé par Dromey (1995) est évidemment basé sur l'architecture qu'il préconise. Comme nous l'avons déjà vu à l'annexe A, cette architecture est constituée de trois entités principales :

- un ensemble de composantes ;
- un ensemble d'attributs qui transportent certaines caractéristiques de la qualité ;
- un ensemble d'attributs de la qualité de haut niveau.

Pour adapter cette architecture au logiciel, le terme composante est remplacé par forme structurelle. Bien que l'architecture puisse être utilisée dans plusieurs situations, elle est adaptée au code en exploitant la propriété que les programmes sont conçus seulement en utilisant des formes structurelles. Une des caractéristiques de ces formes structurelles est qu'elles possèdent chacune un ensemble d'attributs qui transportent certaines caractéristiques de la qualité. Ainsi, ce modèle forme structurelle-propriété facilite l'implantation de qualité dans le logiciel en détectant et classifiant les manquements à la qualité logicielle.

Il y a deux catégories principales de formes structurelles : computationnelle et représentationelle. Il est à noter que les formes structurelles varient d'un langage à l'autre. Ainsi, pour cette étude, le langage *C* sert de référence.

En terme général, les propriétés associées aux formes structurelles qui ont un impact sur la qualité logicielle impliquent deux aspects fondamentaux : *justesse* et *style*. Les propriétés de justesse utilisées couvrent les caractéristiques qui affectent les exigences minimum pour la justesse sans tenir compte du problème solutionné, donc qui sont indépendantes des spécifications du problème. Les propriétés de style couvrent les propriétés associées à la fois au design de haut et de bas niveau et aussi l'ampleur à laquelle la fonctionnalité, à tous les niveaux, est spécifiée, décrite, caractérisée et documentée.

Dans cet ordre d'idées, il est commode de définir les propriétés transportant certaines caractéristiques de la qualité en quatre catégories qui sont les suivantes :

- propriétés de justesse (exigences génériques minimales pour la justesse) ;
- propriétés structurelles (bas niveau, design intra-module) ;
- propriétés modulaires (haut niveau, design inter-module) ;
- propriétés descriptives (formes diverses de spécification/documentation).

Il est à noter que l'ordre d'apparition des catégories correspond à leur importance relative, la catégorie la plus importante étant la première décrite. Le détail de chacune des formes structurelles est élaboré dans le Tableau 2 .3.

Tableau 2 .3 : formes structurelles

Computationnelle	Représentationnelle
Système (ensemble de programmes) Librairie (ensemble d'ADT, fonctions et procédures) Méta-programme (« shell script » qui utilise un programme E/S) Interfaces usagers Objets (ADT) Module (fonctions et procédures) Séquence Énoncé : • boucle ; • sélection ; • appel à une fonction ou procédure ; • assignation. Garde Expression	Records Variables Constantes Types

La prochaine étape, qui est probablement la plus difficile et qui demeure un sujet ouvert, est d'identifier un ensemble de propriétés qui couvrent adéquatement les quatre catégories de formes structurelles. Le Tableau 2 .4 présente chacune de ces propriétés en les classant dans la catégorie appropriée et en mentionnant sur quelles formes structurelles elles ont un impact.

Tableau 2 .4 : propriétés pour les formes structurelles

	CALCULABLE	COMPLET	ASSIGNÉ	PRÉCIS	INITIALISÉ	PROGRESSIF	VARIANT	CONSISTANT	STRUCTURÉ	RÉSOLU	HOMOGÈNE	EFFICACE	NON.REDONDANT	DIRECT	AJUSTABLE	PLAGE.VARIABLE	UTILISÉ	PARAMÉTRISÉ	FAIBLE.COUPLAGE	ENCAPSULÉ	COHÉSIF	GÉNÉRIQUE	ABSTRAIT	SPÉCIFIÉ	DOCUMENTÉ	AUTO-DESCRIPTIF
[illegible]		X															X						X	X	X	X
[illegible]		X				X		X			X						X	X	X			X	X	X	X	X
[illegible]								X	X			X									X			X	X	
[illegible]		X			X	X		X		X	X	X				X								X	X	
[illegible]		X						X				X														
[illegible]								X				X			X										X	
[illegible]		X										X														
[illegible]									X				X													
[illegible]	X						X	X	X			X		X	X											
[illegible]								X									X								X	X
[illegible]			X	X				X						X		X	X			X					X	X
[illegible]				X										X			X			X					X	X
[illegible]														X		X	X			X					X	X

Pour ce qui est des attributs de haut niveau de la qualité, Dromey a choisi d'utiliser ceux du standard ISO 9126 (1991). Il a cependant ajouté l'attribut de *réutilisabilité* qui, selon lui, est tout aussi important que les autres attributs de haut niveau. Le Tableau 2 .5 présente l'impact des propriétés transportant certaines caractéristiques de la qualité sur les attributs de haut niveau.

Tableau 2 .5 : relations entre les attributs de haut niveau et les propriétés

	Produit																									
	Justesse								Structurelle									Modulaire						Descriptive		
Attributs de qualité	CALCULABLE	COMPLET	ASSIGNÉ	PRÉCIS	INITIALISÉ	PROGRESSIF	VARIANT	CONSISTANT	STRUCTURÉ	RÉSOLU	HOMOGÈNE	EFFICACE	NON.REDONDANT	DIRECT	AJUSTABLE	PLAGE.VARIABLE	UTILISÉ	PARAMÉTRISÉ	FAIBLE.COUPLAGE	ENCAPSULÉ	COHÉSIF	GÉNÉRIQUE	ABSTRAIT	SPÉCIFIÉ	DOCUMENTÉ	AUTO-DESCRIPTIF
Fonctionnalité	X	X	X	X	X	X	X	X	X															X		
Fiabilité	X	X	X	X	X	X	X	X	X										X	X				X		
Utilisabilité		X						X				X												X	X	X
Efficacité										X		X	X	X			X									
Maintenabilité		X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Portabilité								X							X			X	X	X	X	X		X	X	X
Réutilisabilité								X							X	X		X	X	X	X	X	X	X	X	X

La Figure 2 .4 présente le modèle de Dromey sous la forme d'une forêt.

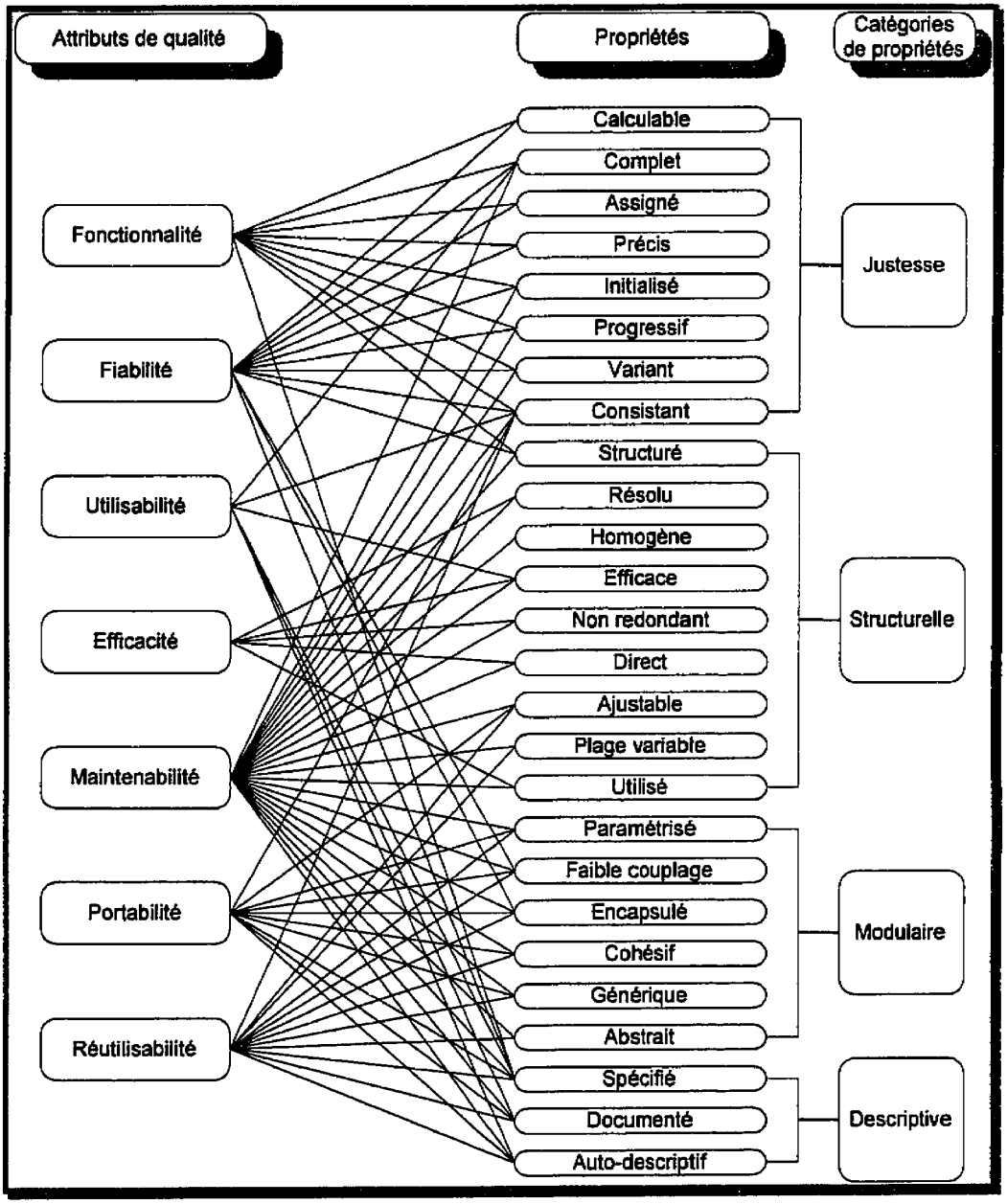


Figure 2 .4 : modèle de Dromey

2 .6. Modèle de McCall, Richards et Walters

Forêt

McCall et al. (1977) ont proposé une catégorisation des facteurs affectant la qualité logicielle. Ces facteurs (appelés « SQF - Software Quality Factors ») sont concentrés sur trois aspects importants des produits logiciels : leurs caractéristiques opérationnelles, leur habileté à subir des changements et leur adaptabilité à de nouveaux environnements. La Figure 2 .5 présente ces facteurs.

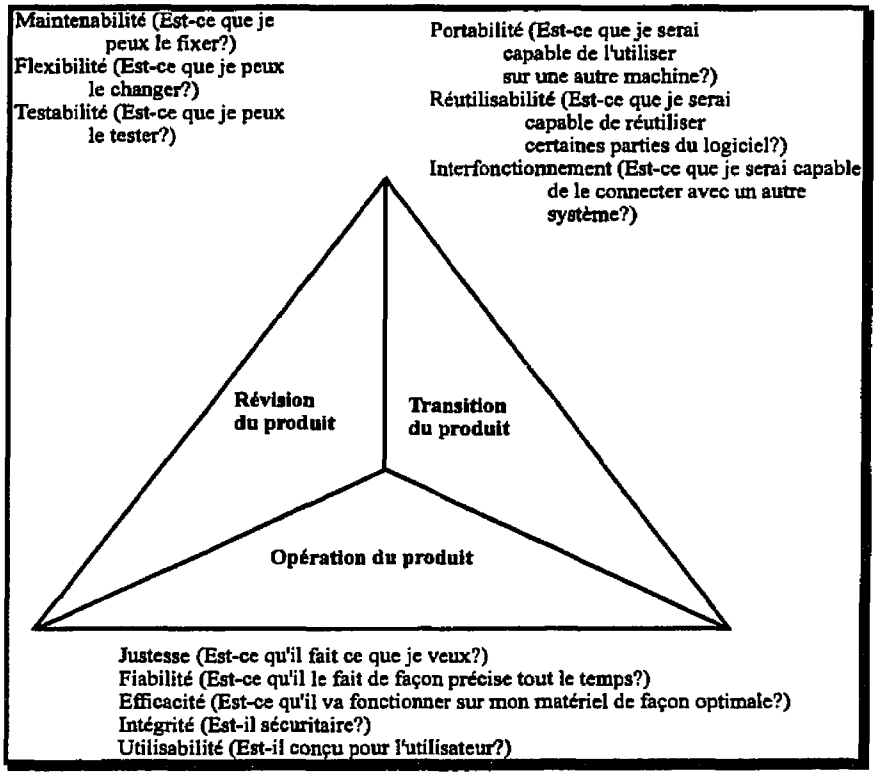


Figure 2 .5 : facteurs de qualité du modèle de McCall et al.

Le Tableau 2 .6 présente la définition de chacun des facteurs du modèle.

Tableau 2 .6 : facteurs de qualité du modèle de McCall et al.

[illegible]	[illegible]
Justesse	Le degré d'accomplissement d'un programme en regard à ses spécifications et l'atteinte des objectifs du client
Fiabilité	Le degré d'accomplissement d'un programme en regard à l'exécution de ses fonctions avec un niveau de précision suffisant
Efficacité	La quantité de ressources de calcul et de codes nécessaires pour un programme pour qu'il soit en mesure d'exécuter ses fonctions
Intégrité	Le niveau de contrôle à l'accès au logiciel ou données par des personnes non autorisées
Utilisabilité	L'effort requis pour comprendre, opérer, préparer les entrées et interpréter les sorties d'un programme
Maintenabilité	L'effort requis pour localiser et corriger une erreur dans un programme
Flexibilité	L'effort requis pour modifier un programme opérationnel
Testabilité	L'effort requis pour tester un programme pour s'assurer qu'il exécute ses fonctions
Portabilité	L'effort requis pour transférer un programme d'un environnement à un autre (matériel ou système logiciel)
Réutilisabilité	Le degré de réutilisation d'un programme (ou partie d'un programme) dans d'autres applications
Interfonctionnement	L'effort requis pour juxtaposer un système à un autre

McCall et al. décomposent les facteurs en un ensemble de supposées métriques. Or, selon notre terminologie, ces métriques ressemblent plus à des critères de qualité qu'à des métriques logicielles puisqu'elles sont d'un niveau d'abstraction encore trop élevé pour correspondre à des caractéristiques d'un produit. Ainsi, pour la suite de la présentation du modèle, nous adapterons la terminologie de McCall et al. en interchangeant le terme métrique par critère.

L'ensemble des critères est donc défini et utilisé pour développer des expressions pour chacun des facteurs selon la relation suivante :

$F_q = c_1 \times m_1 + c_2 \times m_2 + ... + c_n \times m_n$

F_q : facteur de qualité logicielle
c_n : coefficient de régression
m_n : critère qui affecte la qualité logicielle

Le schéma de catégorisation proposé est une échelle de 0 (bas) à 10 (haut). Les critères qui sont utilisés dans le schéma apparaissent dans le Tableau 2 .7.

Tableau 2 .7 : critères de qualité du modèle de McCall et al.

[illegible]	[illegible]
Faculté d'audition	L'aisance selon laquelle la conformité aux standards peut être vérifiée
Précision	La précision des calculs et du contrôle
Similitude des communications	Le degré selon lequel les interfaces standards, les protocoles et la largeur de bande sont utilisés
Complétude	Le degré selon lequel l'implantation complète des fonctions requises a été atteinte
Complexité	
Concision	La compacité d'un programme en terme de ligne de code
Consistance	L'utilisation de design uniforme et de documentation technique tout au long du projet de développement logiciel
Similitude des données	L'utilisation de structures de données et de types standards tout au long du programme
Tolérance aux erreurs	Les dommages qui surviennent lorsque le programme rencontre un erreur
Efficacité de l'exécution	Les performances au niveau de la durée d'exécution du programme
Extensibilité	Le degré selon lequel le design architectural, de données ou procédurel, peut être étendu
Généralité	L'ampleur des applications potentielles des composantes du programme
Indépendance au matériel	Le degré selon lequel le logiciel est découplé du matériel sur lequel il opère
Instrumentation	Le degré selon lequel le programme surveille ses propres opérations et identifie les erreurs qui surviennent
Modularité	L'indépendance fonctionnelle des composantes du programme
Faculté d'opération	La facilité d'opération d'un programme
Sécurité	La disponibilité de mécanismes qui contrôlent ou protègent le programme et les données
Auto-descriptif	Le degré selon lequel le code source fournit une documentation adéquate
Simplicité	Le degré selon lequel un programme peut être compris sans difficulté

Critère de qualité	Définition
Indépendance aux systèmes logiciels	Le degré selon lequel le programme est indépendant des caractéristiques des langages de programmation non standard, des caractéristiques du système d'opération et des contraintes des autres environnements
Faculté de traçage	L'habileté à suivre la trace d'une représentation d'un design ou des composantes actuelles d'un programme jusqu'aux exigences initiales
Formation	Le degré selon lequel le logiciel assiste les nouveaux utilisateurs à appliquer le nouveau système

Les relations entre les facteurs de qualité logicielle et les critères apparaissent dans la Figure 2 .6. Le poids accordé à chacun des critères est dépendant des produits locaux et des préoccupations locales.

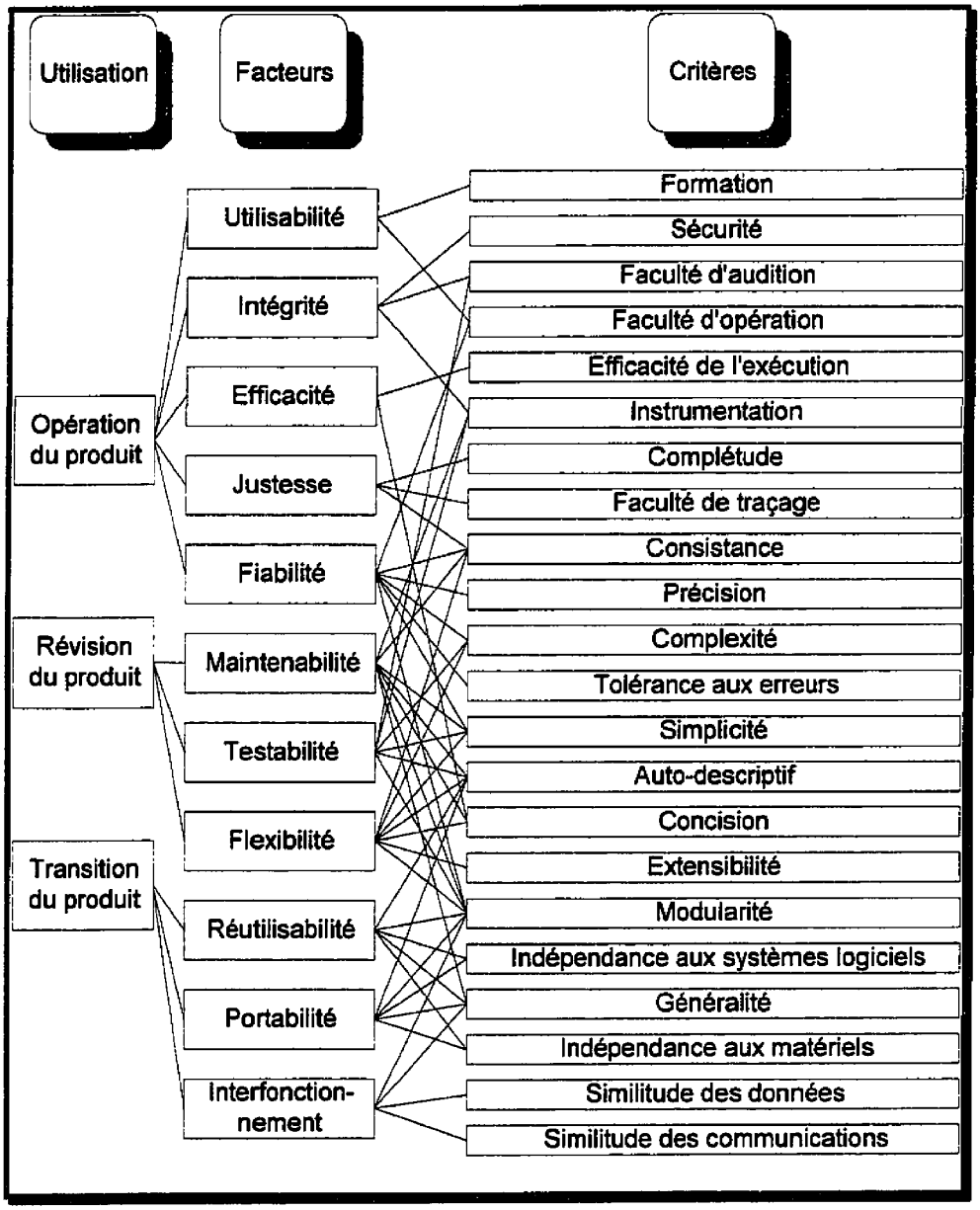


Figure 2 .6 : modèle de McCall et al.

2 .7. Modèle de ISO 9126

N arborescences

Ce standard international a été instauré pour contrer principalement deux problèmes : premièrement, le nombre de modèles de qualité logicielle publiés cause une confusion à savoir en quoi consiste vraiment un logiciel de qualité ; deuxièmement les termes de qualité employés ne sont pas définis et sont interprétés de façon différente par différents experts.

Ce standard définit six caractéristiques qui décrivent, avec un minimum d'entrelacements, la qualité logicielle. Ces caractéristiques fournissent une base pour un raffinement subséquent. Bien que des sous-caractéristiques de chacune des caractéristiques principales soient proposées, elles ne font pas partie du standard. Or, il s'avère que la majorité de la communauté informatique s'entend pour dire que ces sous-caractéristiques semblent former un ensemble assez complet. Le standard ne fournit pas de métriques, de méthodes de mesure, de classements et d'évaluations. Ce standard adhère à la définition de la qualité de ISO8402.

La définition des caractéristiques et le modèle d'évaluation de la qualité associés au standard sont applicables lors de la spécification des exigences et pour évaluer la qualité des produits logiciels durant leur cycle de vie. Les caractéristiques peuvent être applicables à n'importe quel type de logiciel. Ce standard vise les personnes associées à l'acquisition, au développement, à l'utilisation, au support, à la maintenance et à l'audition de logiciels.

La Figure 2 .7 présente les caractéristiques et les sous-caractéristiques du standard ISO-9126.

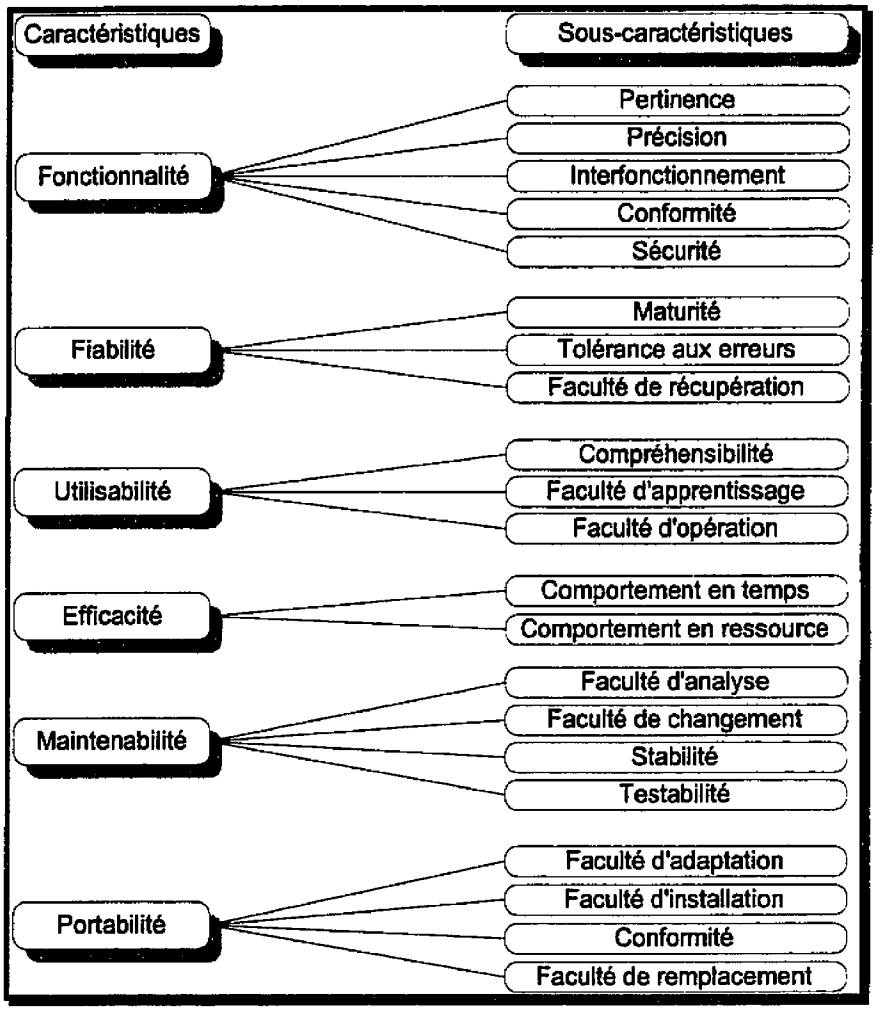


Figure 2 .7 : modèle de ISO-9126

Les définitions des caractéristiques et des sous-caractéristiques sont données dans le Tableau 2 .8 et le Tableau 2 .9 respectivement.

Tableau 2 .8 : définitions des caractéristiques du standard ISO-9126

Caractéristique	Définition
Fonctionnalité	Un ensemble d'attributs qui porte sur l'existence d'un ensemble de fonctions et de leurs propriétés. Les fonctions sont celles qui satisfont les besoins explicites et implicites
Fiabilité	Un ensemble d'attributs qui porte sur la capacité du logiciel à maintenir son niveau de performance sous des conditions et une durée spécifiées
Utilisabilité	Un ensemble d'attributs qui porte sur l'effort nécessaire pour l'utilisation et sur l'évaluation individuelle d'une telle utilisation par les utilisateurs
Efficacité	Un ensemble d'attributs qui porte sur la relation entre le niveau de performance du logiciel et la quantité de ressources utilisées sous des conditions spécifiées
Maintenabilité	Un ensemble d'attributs qui porte sur l'effort nécessaire pour faire les modifications spécifiées
Portabilité	Un ensemble d'attributs qui porte sur l'habileté du logiciel à être transféré d'un environnement à un autre

Tableau 2 .9 : définitions des sous-caractéristiques du standard ISO-9126

Caractéristique	Sous-caractéristique	Définition
Fonctionnalité	Pertinence	Attributs du logiciel qui portent sur la présence et la pertinence d'un ensemble de fonctions pour une certaine tâche
	Précision	Attributs du logiciel qui portent sur l'exactitude des résultats ou des effets
	Interfonctionnement	Attributs du logiciel qui portent sur son habileté à interagir avec certains systèmes
	Conformité	Attributs du logiciel qui font adhérer le logiciel aux standards, conventions ou régulations applicables dans les lois et prescriptions similaires
	Sécurité	Attributs du logiciel qui portent sur son habileté à prévenir l'accès non autorisé, soit accidentel ou délibéré aux programmes et données
Fiabilité	Maturité	Attributs du logiciel qui portent sur la fréquence des défaillances à cause des fautes dans le logiciel

	Tolérance aux erreurs	Attributs du logiciel qui portent sur son habileté à maintenir un certain niveau de performance dans le cas de fautes logicielles ou d'infractions au niveau de l'interface
	Faculté de récupération	Attributs du logiciel qui portent sur la capacité de rétablir son niveau de performance et récupérer les données directement affectées dans le cas de défaillances et sur le temps et l'effort nécessaires pour y arriver.
Utilisabilité	Compréhensibilité	Attributs du logiciel qui portent sur l'effort de la part des utilisateurs pour reconnaître le concept logique et son applicabilité
	Faculté d'apprentissage	Attributs du logiciel qui portent sur l'effort de la part des utilisateurs pour apprendre son application (e.g., contrôle des opérations, entrées, sorties)
	Faculté d'opération	Attributs du logiciel qui portent sur l'effort de la part des utilisateurs pour l'opération et le contrôle des opérations
Efficacité	Comportement en temps	Attributs du logiciel qui portent sur le temps de réponse et de traitement et sur le taux de traitement dans l'exécution de ses fonctions
	Comportement en ressources	Attributs du logiciel qui portent sur la quantité de ressources utilisées et sur la durée d'une telle utilisation dans l'exécution de ses fonctions
Maintenabilité	Faculté d'analyse	Attributs du logiciel qui portent sur l'effort nécessaire pour diagnostiquer les déficiences ou les causes de défaillances ou pour l'identification des parties à être modifiées
	Faculté de changement	Attributs du logiciel qui portent sur l'effort nécessaire pour les modifications, l'enlèvement des fautes ou des changements d'environnement
	Stabilité	Attributs du logiciel qui portent sur le risque d'effets non prévus des modifications
	Testabilité	Attributs du logiciel qui portent sur l'effort nécessaire pour valider le logiciel modifié
Portabilité	Adaptabilité	Attributs du logiciel qui portent sur l'opportunité de l'adapter à différents environnements sans appliquer d'autres actions ou moyens autres que ceux requis pour cet objectif

	Faculté d'installation	Attributs du logiciel qui portent sur l'effort nécessaire pour installer le logiciel dans un certain environnement
	Conformité	Attributs du logiciel qui font adhérer le logiciel aux standards et conventions reliés à la portabilité
	Faculté de remplacement	Attributs du logiciel qui portent sur l'opportunité et l'effort de l'utiliser à la place d'un autre logiciel dans l'environnement de ce logiciel

2.7.1. Guide pour l'utilisation des caractéristiques de qualité

2.7.1.1. Usage

Ce standard est applicable pour définir les exigences de qualité logicielle et évaluer les produits logiciels. Les organisations devront elles-mêmes établir leur propre programme de mesure puisqu'il n'y a pas assez de métriques qui font l'objet d'un consensus. Le standard fournit un modèle de processus d'évaluation logicielle qui s'avère pertinent, modèle qui est l'objet de la sous-section suivante.

2.7.1.2. Modèle du processus d'évaluation

La Figure 2.8 présente les principales étapes requises pour l'évaluation de la qualité logicielle. En raison de la nature de haut niveau de la Figure 2.8, certaines procédures telles l'analyse et la validation des métriques ne sont pas présentées. Le processus se déroule en trois stades : premièrement, il faut définir les exigences de qualité ; deuxièmement il faut préparer l'évaluation et finalement il faut exécuter l'évaluation proprement dite. Ce processus peut être appliqué dans chacune des phases du cycle de vie pour chaque composante du produit logiciel.

Définition des exigences de qualité

L'objectif du stade initial est de spécifier les exigences en termes de caractéristiques et de sous-caractéristiques de la qualité. Les exigences expriment les demandes de l'environnement pour le produit logiciel sous considération et se doivent d'être définies avant le développement. Comme un produit logiciel est décomposé en composantes majeures, les exigences dérivées à partir du produit global peuvent différer de celles des différentes composantes.

Préparation de l'évaluation

L'objectif du deuxième stade est de préparer les bases pour l'évaluation.

Sélection des métriques de qualité

Dans la majorité des cas, la façon dont les caractéristiques ont été définies ne permet pas de les mesurer directement. Il est donc nécessaire d'établir des métriques qui corrèlent aux caractéristiques du produit logiciel. Chaque caractéristique quantifiable du logiciel et chaque interaction quantifiable du logiciel avec son environnement qui corrèlent avec une caractéristique peuvent être établies comme une métrique.

Définition des niveaux de classement

Les caractéristiques quantifiables peuvent être mesurées quantitativement en utilisant des métriques de qualité. Le résultat, la valeur mesurée, est transporté sur une échelle. Cette valeur ne démontre pas le niveau de satisfaction. Pour cet objectif, les échelles doivent être divisées en classes correspondant aux différents degrés de satisfaction vis-à-vis les exigences (voir Figure 2 .9). Puisque la qualité réfère aux besoins, il n'est pas possible de définir des niveaux de classement généraux. Ils doivent donc être définis pour chaque évaluation spécifique.

Définition des critères d'évaluation

Pour évaluer la qualité d'un produit, les résultats de l'évaluation des différentes caractéristiques doivent être agrégés. L'évaluateur doit préparer une procédure pour ceci en utilisant, par exemple, des tables de décisions ou des moyennes pondérées. La procédure inclura généralement d'autres aspects tels le temps et le coût qui contribuent à l'évaluation de la qualité des produits logiciels dans un environnement particulier.

Procédure d'évaluation

Le dernier stade du processus d'évaluation est raffiné en trois étapes : mesure, classement et évaluation.

Mesure

Pour la mesure, les métriques sélectionnées sont appliquées au produit logiciel. Les résultats sont des valeurs sur les échelles des métriques.

Classement

Dans l'étape de classement, le niveau de classement est déterminé pour une valeur mesurée (voir Figure 2 .9).

Évaluation

L'évaluation est l'étape finale du processus d'évaluation logicielle où un ensemble de niveaux de classement est agrégé. Le résultat est un énoncé de la qualité du produit logiciel. Ensuite, la qualité agrégée est comparée avec d'autres aspects tels le temps et le coût. Finalement, les décisions de gestion seront faites en se basant sur les critères de gestion. Le résultat est une décision de gestion sur l'acceptation ou le rejet ou sur la livraison ou non livraison du produit logiciel.

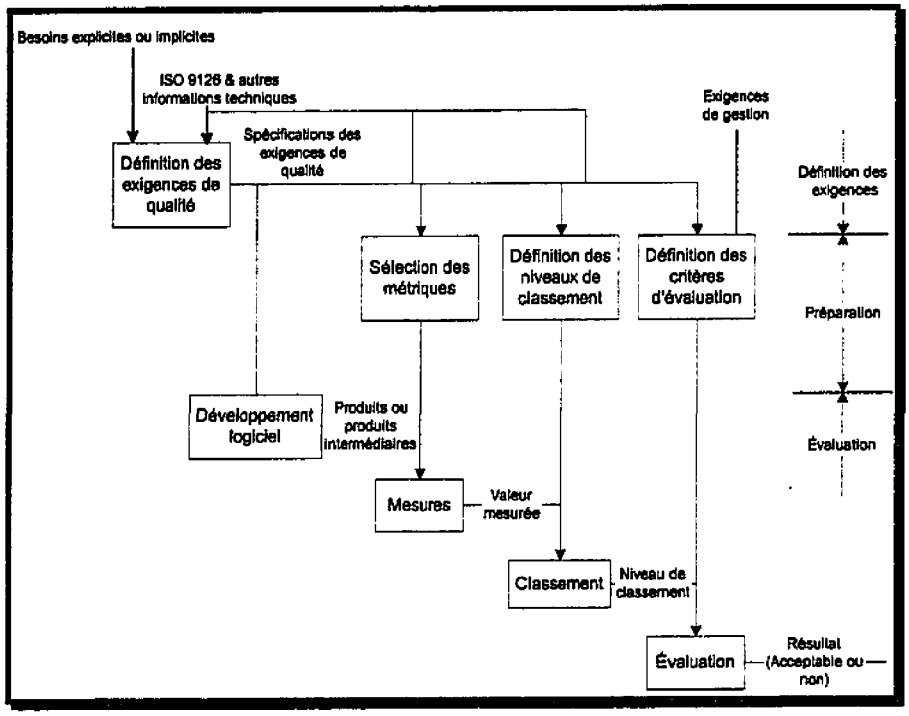


Figure 2 .8 : modèle du processus d'évaluation

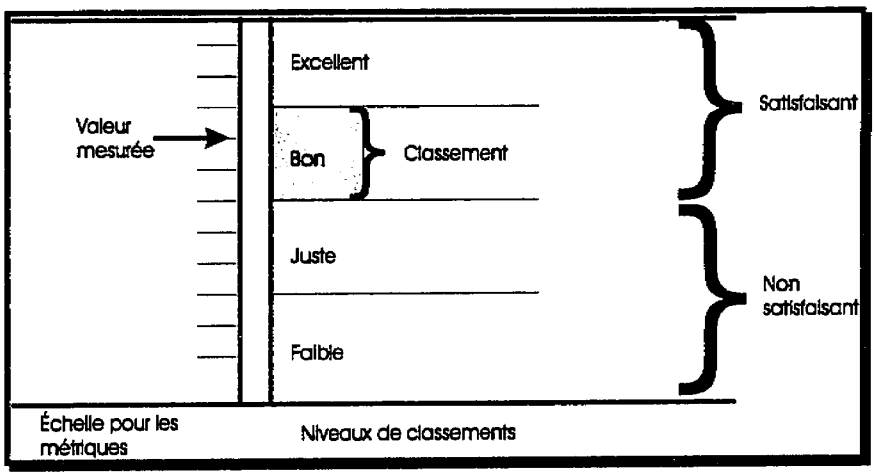


Figure 2 .9 : Valeur mesurée et classement résultant

2 .7.2. Critique du modèle ISO 9126

Selon Dromey (1995), bien que ce standard fournisse un guide de haut niveau, il ne va pas assez loin pour supporter l'ajout de qualité dans le logiciel.

Encore une fois au dire de Dromey (1995), il y a une omission majeure au standard qui consiste en un manque d'emphase sur la *réutilisabilité*. Ainsi, il prétend que la réutilisabilité est un attribut de haut niveau de la qualité qui mérite un statut similaire aux six autres attributs de haut niveau. Une autre raison, qui prône en faveur de l'ajout de cet attribut, est que cela va encourager les responsables du développement logiciel à porter une attention particulière à concevoir des logiciels qui sont réutilisables.

2 .8. Modèle de IEEE std. 1061-1992

N arborescences

Le modèle qui est présenté ne fait pas partie du standard IEEE std. 1061-1992 (1992) même s'il est inclus dans sa description. Ainsi, il est décrit simplement à but informatif.

Le Tableau 2 .10 et le Tableau 2 .11 illustrent les descriptions des facteurs et sous-facteurs qui composent le modèle.

Tableau 2 .10 : facteurs du modèle de IEEE std. 1061-1992

Facteur	Définition
Efficacité	Un attribut qui porte sur la relation entre le niveau de performance et la quantité de ressources utilisées sous certaines conditions
Fonctionnalité	Un attribut qui porte sur l'existence de certaines propriétés et fonctions qui satisfont les besoins implicites ou explicites des utilisateurs
Maintenabilité	Un attribut qui porte sur l'effort nécessaire pour des modifications spécifiques
Portabilité	Un attribut qui porte sur l'habileté d'un logiciel à être transféré d'un environnement à un autre
Fiabilité	Un attribut qui porte sur la capacité d'un logiciel de maintenir son niveau de performance sous certaines conditions durant une certaine période de temps
Utilisabilité	Un attribut qui porte sur l'effort nécessaire pour l'utilisation (incluant la préparation pour l'utilisation et l'évaluation des résultats) et sur l'évaluation d'une telle utilisation par les utilisateurs

Tableau 2 .11 : sous-facteurs du modèle IEEE std. 1061-1992

Facteur	Sous-facteur	Définition
Efficacité	Économie en temps	Capacité d'un logiciel d'exécuter les fonctions spécifiées sous des conditions explicites ou implicites dans des tranches de temps appropriées
	Économie en ressources	Capacité d'un logiciel d'exécuter les fonctions spécifiées sous des conditions explicites ou implicites en utilisant la quantité de ressources appropriée

Fonctionnalité	Complétude	Le degré selon lequel un logiciel possède les fonctions nécessaires et suffisantes pour satisfaire les besoins des utilisateurs
	Justesse	Le degré selon lequel toutes les fonctions logicielles sont spécifiées
	Sécurité	Le degré selon lequel un logiciel peut détecter des fuites d'informations, des pertes d'informations, des utilisations illégales et des destructions des ressources du système
	Compatibilité	Le degré selon lequel un nouveau logiciel peut être installé sans changer l'environnement et les conditions qui avaient été préparés pour le logiciel qui sera remplacé
	Interfonctionnement	Le degré selon lequel le logiciel peut être facilement connecté avec d'autre systèmes
Maintenabilité	Faculté de correction	L'effort requis pour corriger des erreurs dans le logiciel et s'occuper des plaintes des usagers
	Extensibilité	L'effort requis pour améliorer ou modifier l'efficacité ou les fonctions du logiciel
	Testabilité	L'effort requis pour tester le logiciel
Portabilité	Indépendance aux matériels	Le degré selon lequel le logiciel ne dépend pas d'environnements matériels spécifiques
	Indépendance aux systèmes logiciels	Le degré selon lequel le logiciel ne dépend pas d'environnements logiciels spécifiques
	Faculté d'installation	L'effort requis pour ajuster le logiciel dans un nouvel environnement
	Réutilisabilité	Le degré selon lequel le logiciel peut être réutilisé dans des applications différentes de l'originale
Fiabilité	Non déficience	Le degré selon lequel le logiciel ne contient pas d'erreurs non détectées
	Tolérance aux erreurs	Le degré selon lequel le logiciel continuera à fonctionner sans une défaillance qui causerait des torts aux utilisateurs. Également, le degré selon lequel le logiciel inclut des opérations de dégradation et des fonctions de récupération
	Disponibilité	Le degré selon lequel le logiciel demeure opérable en présence de défaillances du système
Utilisabilité	Compréhensibilité	L'effort requis par l'utilisateur pour comprendre le logiciel
	Facilité d'apprentissage	Le degré selon lequel l'effort requis par l'utilisateur pour comprendre le logiciel est minimisé

	Faculté d'opération	Le degré selon lequel les opérations du logiciel correspondent aux objectifs, à l'environnement et aux caractéristiques physiologiques des utilisateurs, incluant les facteurs d'ergonomie tels la couleur, la forme, le son, etc.
	Faculté de communication	Le degré selon lequel le logiciel est conçu en accord avec les caractéristiques psychologiques des utilisateurs

La Figure 2 .10 présente le modèle de IEEE std. 1061-1992 sous le format de N arborescences.

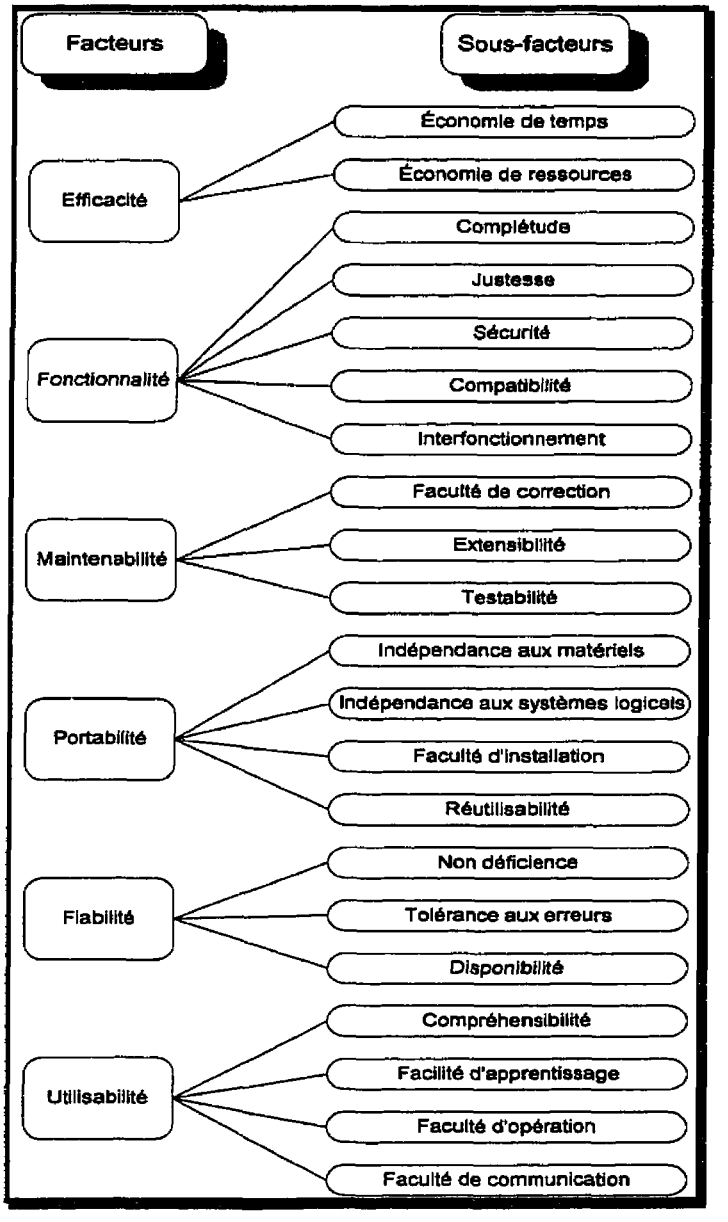


Figure 2 .10 : modèle de IEEE std. 1061-1992

Le Tableau 2 .12 présente certaines métriques importantes qui peuvent servir pour mesurer des facteurs ou sous-facteurs.

Tableau 2 .12 : métriques du modèle IEEE std. 1061-1992

Complétude	Ratio du nombre de documents complets ou composantes logicielles sur le nombre total de documents ou composantes logicielles
Consistance	Ratio du nombre de documents ou composantes logicielles qui sont sans contradictions sur le nombre total de documents ou de composantes logicielles
Justesse	Ratio du nombre d'objectifs des utilisateurs qui ont été correctement implantés dans les composantes logicielles sur le nombre total d'objectifs des utilisateurs
Interfonctionnement	Ratio du nombre de spécifications d'interfaces ou composantes logicielles dont les interfaces sont compatibles sur le nombre total de spécifications d'interfaces ou de composantes logicielles
Maintenabilité	Ratio du temps total de correction sur le nombre total de corrections
Fiabilité	Ratio du nombre d'utilisations qui ont produit des sorties adéquates sur le nombre total d'utilisations
Testabilité	Identification des chemins indépendants correspondant à la métrique de McCabe pour le testage des chemins
Faculté de traçage	Ratio du nombre de documents ou de composantes logicielles pour une phase qui peuvent être reliés à la phase précédente sur le nombre total de documents ou de composantes logicielles pour la phase

2 .9. Modèle de Hewlett-Packard

N arborescences

Grady et Caswell (1987) de Hewlett-Packard ont développé un ensemble de facteurs de la qualité logicielle portant le sigle FURPS (« Functionality, Usability, Reliability, Performance and Supportability»). Les définitions données de ces facteurs sont les suivantes :

- La *fonctionnalité* est estimée en évaluant l'ensemble de caractéristiques et les capacités du programme, la généralité des fonctions qui sont livrées et la sécurité du système entier ;

- L'*utilisabilité* est estimée en considérant le facteur humain, l'esthétique générale, la consistance et la documentation ;

- La *fiabilité* est estimée en mesurant la fréquence et la sévérité des défaillances, la précision des résultats en sorties, le temps entre deux défaillances, l'habileté à récupérer d'une défaillance et la prévisibilité du programme ;

- La *performance* est mesurée en évaluant le temps de traitement, le temps de réponse, la consommation des ressources, la capacité de traitement et l'efficacité ;

- La *faculté de soutien* combine l'habileté à étendre le programme (extensibilité), faculté d'adaptation, faculté de service (ces trois attributs représentent un terme plus commun - maintenabilité), testabilité, compatibilité, faculté de configuration, facilité d'installation et facilité de localisation des problèmes.

La Figure 2 .11 synthétise tous les facteurs de qualité et les attributs correspondants du modèle de Hewlett-Packard.

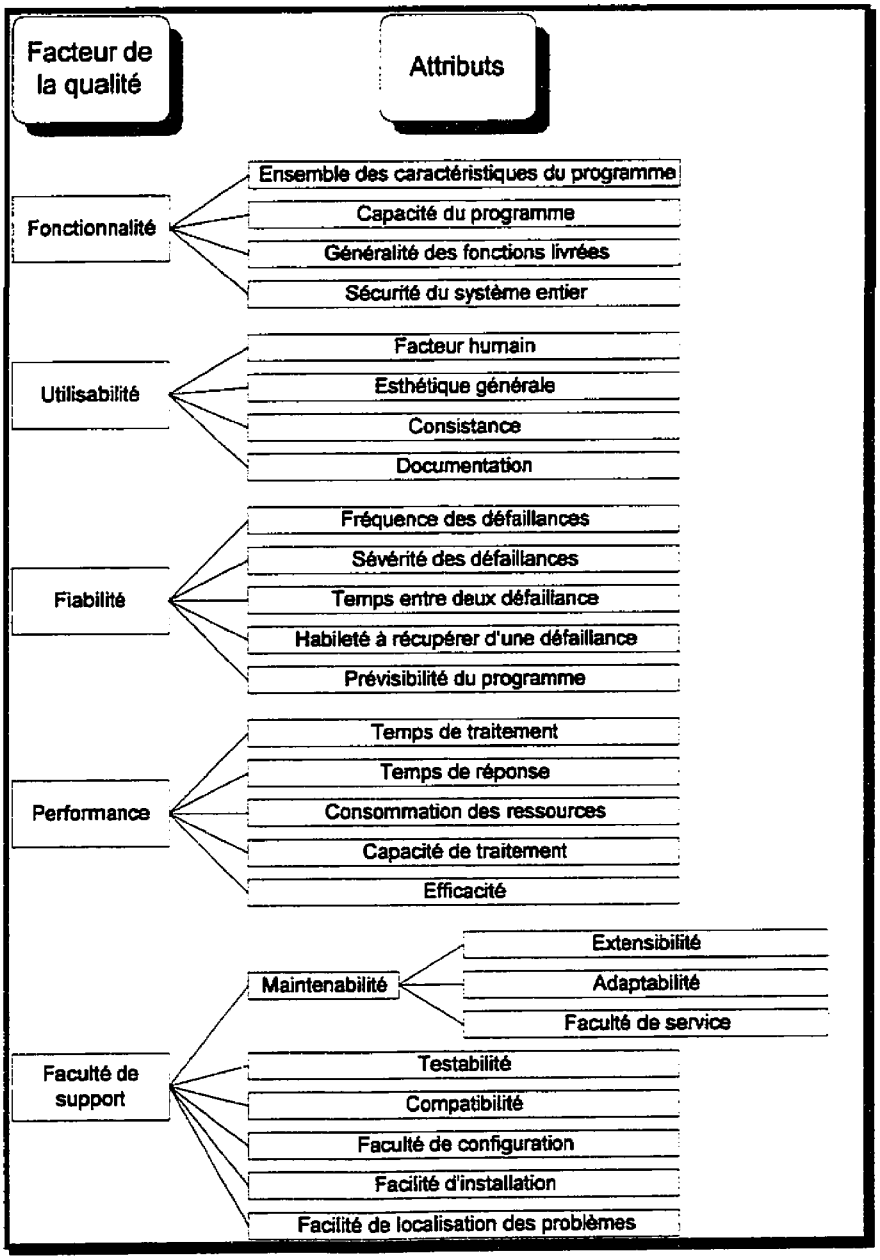


Figure 2 .11 : modèle de Hewlett-Packard

Les facteurs de qualité FURPS et les attributs décrits précédemment peuvent être utilisés pour établir des métriques de qualité à chacune des étapes du processus d'ingénierie logicielle. Ainsi, Grady et Caswell (1987) proposent une matrice (voir Tableau 2 .13) pour guider la collecte des mesures FURPS simples. Étant donné que les termes contenus dans la matrice sont très techniques, nous avons préféré garder la terminologie anglaise originale.

Tableau 2 .13 : guide pour la collecte des mesures FURPS

	Investigation/ Specifications	Design	Implementation	Testing	Support
F	# target users to review spec or prototype % grade on report card from user % features competitive with other products # interfaces with existing products	% spec includes in design # changes to spec due to design requirement # users to review change if needed	% designs included in code # code changes due to omissions discovered % features removed (reviewed by original target user)	% features tested at alpha sites % user documentation tested against product # target alpha customers	# known problem reports sales act. reports (esp. lost sales) users surveys internal HP user surveys
U	# target users to review spec or prototype % grade on documentation plan by target user % grade on usability of prototype	% grade of design as compared to objectives # changes to prototype manuals after review	% grade by other la user % grade by product marketing, documentation % original users to review any change	# changes to product after alpha test % grade from usability lab testing % grade by test sites	# user misunderstandings
R	# omissions noted in reviews of objectives (reliability goals) # changes to project plan, test plan after review	# changes to design after review due to error % grade of design as compared to objectives	% code changed due to reliability errors % code covered by test cases # defects/KNCSS during module testing	MTTF (MTBF) % hrs reliability testing # defects/1K hrs # defects total defects rate before release ckpoints	# known problem reports # defects/KNCSS
P	# changes to objectives after review % grade on objectives by target user % grade on objective by product managers	% product to be modeled defined modeled environment	performance tests achieve % of modeled expectations % of code tested wth targeted performance suite (module)	achieve performance goal with regard to environment(s) tested % of code tested wth targeted performance suite (system)	
S	# changes to support objectives after review by field & CPE	# design changes by CPE & fiel # diagnostic/recovery changes by CPE & field input	MTTR objective (time) MTTC objective (time) time to train tester, use of documentation	MTTR objective (time) MTTC objective (time) time to train tester, use of documentation	MTTR objective (time) MTTC objective (time) time to train tester, use of documentation

2 .10. Modèle de Boloix et Robillard

N arborescences M-aires complètes

Le modèle de Boloix et Robillard (1994A, 1994B, 1995A, 1995B) a pour principal objectif de fournir un schéma de classification pour identifier des ensembles de projets similaires. Dans le même ordre d'idées, le modèle proposé guide les organisations lors de la classification des données provenant de métriques. Également, il peut être utilisé pour déterminer quels types de métriques devraient être utilisées.

Le modèle utilise le processus d'analyse hiérarchique ("AHP – Analytic Hierarchical Process") pour agréger les données de bas niveau vers un ensemble de données de haut niveau.

Le modèle intègre les connaissances des systèmes selon trois perspectives : le projet de production de logiciel, les caractéristiques du système et la contribution du système à l'organisation. Ces perspectives représentent les trois vues possibles d'un système : l'avant (i.e., développement ou amélioration), le pendant (i.e., le produit lui-même) et l'après (i.e., stage de post-évaluation). Le modèle est présenté à la Figure 2 .12. Il est à noter que nous avons considéré seulement les deux premiers niveaux du modèle pour la classification puisque les autres niveaux sont seulement décrits comme des exemples de raffinements possibles. La Figure 2 .12 présente seulement les raffinements qui présentent un intérêt pour notre domaine de recherche.

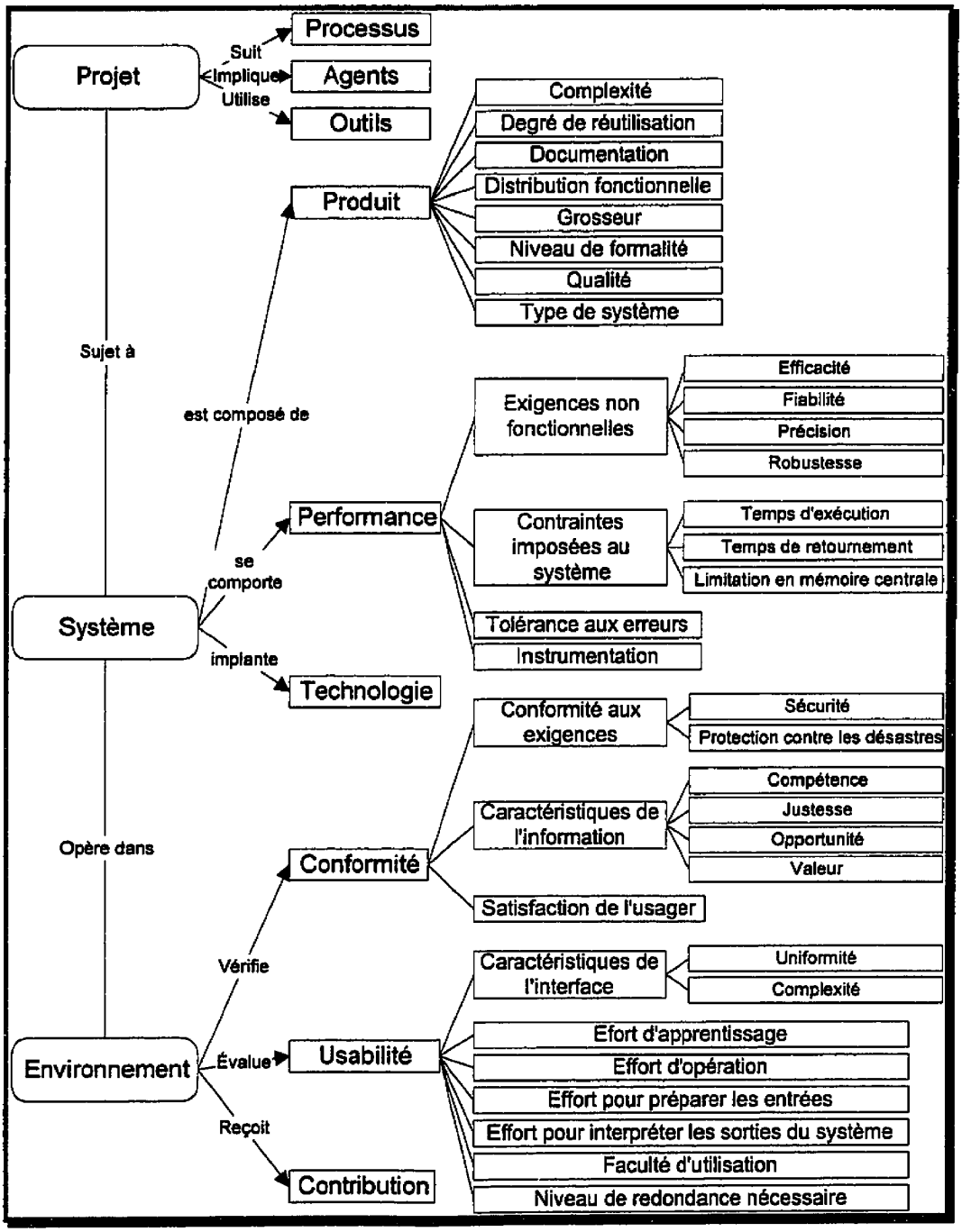


Figure 2 .12 : modèle de Boloix et Robillard

2 .10.1. Approches courantes qui ont servi à l'élaboration du modèle

Selon Boloix et Robillard, beaucoup de chercheurs ont proposé l'utilisation de données de métriques de l'industrie pour effectuer des études multivariables. La plupart de ces recherches ont été effectuées en corrélant les métriques de produits avec l'effort. Cependant, aucune donnée relativement à l'environnement n'est considérée. D'où la conclusion que ces recherches peuvent être trompeuses.

Après avoir analysé les modèles de productivité présentés dans la littérature, Boloix et Robillard ont identifié les principaux domaines qui ont un impact sur la productivité. Le personnel a définitivement un impact majeur, mais sa contribution est difficile à évaluer à cause des critères subjectifs qui la quantifient. Le personnel inclut : utilisateurs, gestionnaires, analystes et programmeurs. Le processus et la méthode de développement définissent l'environnement de production logicielle. Les projets d'envergure suivent généralement un processus strict qui génère des livrables spécifiques qui nécessitent une surcharge de temps plus importante que pour les petits projets. La disponibilité de parties réutilisables améliore la productivité du processus. La structure logicielle a également un impact sur la productivité ; l'architecture logicielle détermine les critères de décomposition généraux. Finalement, le type de technologie disponible pour la production logicielle et l'implantation du système (i.e. la technologie visée) déterminent le niveau d'automatisation du système et de son processus de production.

2 .10.2. Comparaison du modèle avec les autres approches

Le Tableau 2 .14 présente une comparaison du modèle de Boloix et Robillard avec d'autres approches courantes de l'industrie. Les chiffres qui sont présentés dans ce tableau correspondent aux nombres d'attributs qui sont reliés à chacun des attributs du modèle.

Tableau 2 .14 : approches courantes vs modèle de Boloix et Robillard

	WF77	BZ78	B81	M81	H88	CDS86	RB89	KMO91	BBL76	MC79	ISO9126
Processus	9	6	2	6	11	7	3	6			
Agent	5	2	5	4	2	1		24			
Outil	2	5	2	2		4		2			
Produit	9	4	2	19	3	6	1		12	19	10
Performance	2	3	3	2		4			6	8	5
Technologie		1	1	6	1	1					
Conformité	1				1	1			1	2	3
Utilisabilité	1			1					4	4	3
Contribution											

ANNEXE III
CLASSIFICATION DES ATTRIBUTS EXTERNES DE QUALITÉ LOGICIELLE

3.1. Introduction

Dans cette annexe, nous présentons une classification des attributs externes de qualité logicielle. Ces attributs sont issus d'une dizaine de modèles de qualité logicielle existants. La classification est présentée sous la forme de tableau. Le Tableau 3 .1 présente la classification des facteurs de qualité, alors que le Tableau 3 .2 présente quant à lui la classification des critères de qualité. Comme vous le remarquerez, la littérature est très confuse pour ce qui est de la distinction entre un facteur et un critère de qualité. Ainsi, il arrive qu'un facteur d'un modèle corresponde à un critère d'un autre modèle, l'inverse étant également vrai.

Les regroupements dans la classification ne se font pas au niveau des attributs mais au niveau des concepts décrivant les attributs. Ainsi, un même attribut peut se retrouver dans plusieurs groupes différents. Par exemple, l'attribut *maintenabilité* se retrouve dans deux groupes selon que le concept décrivant la *maintenabilité* réfère à l'évolution du logiciel ou simplement à la correction du logiciel.

3 .2. Classification des facteurs de qualité

Tableau 3 .1: classification des facteurs de qualité de la littérature

Capacité d'exécution du logiciel versus la quantité de ressources utilisées		Efficacité	Robillard (1985)	Capacité d'exécution des structures qui composent un programme, sans gaspillage des ressources de la machine (mémoire centrale, mémoire secondaire, canaux de communication et temps d'exécution)
		Efficiency	[A85]	The amount of computing resources required to perform a user-defined function (Does it run on your hardware as well as it can?)
		Efficiency	Boehm et al. (1978)	A software product possesses the characteristic efficiency to the extent that it fulfills its purpose without waste of resources
		Efficiency	Bowen et al. (1985)	Relative extent to which a resource is utilized (i.e., storage space, processing time, communication time)
		Efficiency	Deutsch et Willis (1988)	Efficiency is the ratio of actual versus budgeted resource utilization by the CSCI. Resources include processor time, memory size, and communication bandwidth.
		Efficiency	[EM87]	Relative extent to which a resource is utilized (i.e., storage space, processing time, communication time)
		Efficiency	[G77]	Efficiency is the ratio of useful work performed to the total energy expended
		Efficiency	IEEE1061-1992 (1992)	An attribute that bears on the relationship of the level of performance to the amount of resources used under stated conditions
		Efficiency	McCall et al. (1977)	The amount of computing resources and code required by a program to perform its functions
		Rendement (Efficiency)	ISO-9126 (1991)	Ensemble d'attributs portant sur le rapport existant entre le niveau de service d'un logiciel et la quantité de ressources utilisées, dans des conditions déterminées
		Capacité fonctionnelle (Functionality)	ISO-9126 (1991)	Ensemble d'attributs portant sur l'existence d'un ensemble de fonctions et leurs propriétés données. Les fonctions sont celles qui satisfont aux besoins exprimés ou implicites

Degré selon lequel le logiciel répond aux besoins		Correctness	[A85]	The degree to which a program satisfies the user's specification (Does it do what you want?)
		Correctness	Bowen et al. (1985)	Extend to which the software conforms to its specification and standards
		Correctness	Deutsch et Willis (1988)	Correctness is the degree to which the CSCI satisfies the specified requirements
		Correctness	[EM87]	Extent to which the software conforms to its specifications and standards
		Correctness	McCall et al. (1977)	The extent to which a program satisfies its specification and fulfills the customer's mission objectives
		Functionality	[IEEE1061-1992 (1992)	An attribute that bears on the existence of certain properties and functions that satisfy stated or implied needs of users
Protection des composantes du logiciel à des usages non autorisés		Integrity	[A85]	How well are the software and data protected from security breeches (Is it controlled and secure?)
		Integrity	Bowen et al. (1985)	Extent to which the software will perform without failures due to unauthorized access to the code or data within a specified time period
		Integrity	Deutsch et Willis (1988)	Integrity is the extent to which the CSCI controls access to system resources. Resources include data-base items, functions, and software-controlled hardware.
		Integrity	[EM87]	Extent to which the software will perform without failure due to unauthorized access to the code or data within a specified time period
		Integrity	McCall et al. (1977)	The extent to which access to software or data by unauthorized persons can be controlled
Possibilité d'interfacer le logiciel avec d'autres systèmes		Interoperability	[A85]	How much effort is required to couple this program or system with another? (Will it interface with other systems?)
		Interoperability	Bowen et al. (1985)	Relative effort to couple the software of one system to the software of another system
		Interoperability	Deutsch et Willis (1988)	Interoperability is the extent of effort required to facilitate the interface of this CSCI with other systems or CSCIs
		Interoperability	[EM87]	Relative effort to couple the software of one system to the software of another system
		Interoperability	McCall et al. (1977)	The effort required to couple one system to another

		Maintainability	IEEE1061-1992 (1992)	An attribute that bears on the effort needed for specific modifications
Évolution et correction du logiciel		Maintenabilité (Maintainability)	ISO-9126 (1991)	Ensemble d'attributs portant sur l'effort nécessaire pour faire des modifications données. Une modification peut comprendre des corrections, des améliorations ou des adaptations du logiciel à des changements d'environnement, ou à des exigences et des spécifications fonctionnelles
	Évolution	Expandability	Bowen et al. (1985)	Relative effort to increase the software capability or performance by enhancing current functions or by adding new functions or data
		Expandability	Deutsch et Willis (1988)	Expandability is the extent of effort required to expand (add new) software capabilities or performance
		Expandability	[EM87]	Relative effort to increase the software capability or performance by enhancing current functions or by adding new functions or data
		Flexibilité	Robillard (1985)	Facilité avec laquelle on peut modifier correctement un programme
		Flexibility	[A85]	How much effort does it take to enhance the program? (Can you change it?)
		Flexibility	Bowen et al. (1985)	Ease of effort for changing the software missions, functions, or data to satisfy other requirements
		Flexibility	Deutsch et Willis (1988)	Flexibility is the extent of effort required to change (modify existing) software to accommodate changes in requirements
		Flexibility	[EM87]	Ease of effort for changing the software missions, functions, or data to satisfy other requirements
		Flexibility (Adaptability)	[G77]	Open-ended flexibility is a measure of the ease with which new functions can be added to a system
		Flexibility	McCall et al. (1977)	The effort required to modify an operational program
		Maintainability	Boehm et al. (1978)	A software product possesses the characteristic maintainability to the extent that it facilitates updating to satisfy new requirements
	Correction	Maintainability	[A85]	How much effort will be required to locate and repair errors in the program? (Can you fix it?)
		Maintainability	Bowen et al. (1985)	Ease of effort for locating and fixing a software failure within a specified period of time
		Maintainability	[EM87]	Ease of effort for locating and fixing a software failure within a specified time period

Critère	Sous-critère	Appellation	Auteur	Définition
		Maintainability	[G77]	The probability that, when maintenance action is initiated under stated conditions, a failed system (s) will be restored to operable condition within a specified time (t)
		Maintainability	McCall et al. (1977)	The effort required to locate and fix an error in a program
		Maintainability	Deutsch et Willis (1988)	Maintainability is the extent of effort required to find and fix errors in the CSCI
Adaptation du logiciel à un nouvel environnement		Portabilité (Portability)	ISO-9126 (1991)	Ensemble d'attributs portant sur l'aptitude du logiciel à être transféré d'un environnement à un autre. L'environnement peut être organisationnel, matériel ou logiciel
		Portabilité	Robillard (1985)	Facilité avec laquelle un programme peut s'adapter à un nouvel environnement.
		Portability	[A85]	What kind of effort will it take to transfer a program from one machine to another? (Will it run on your micros, minis, and mainframes?)
		Portability	Boehm et al. (1978)	A software product possesses the characteristic portability to the extent that it can be operated easily and well on computer configurations other than its current one
		Portability	Bowen et al. (1985)	Relative effort to transport the software for use in another environment (hardware, configuration and/or software system environment)
		Portability	Deutsch et Willis (1988)	Portability is the extent of effort required to transfer this CSCI from one hardware or software system environment to another
		Portability	[EM87]	Relative effort to transport the software for use in another environment (hardware configuration and/or software system environment)
		Portability	[G77]	Portability is a reflection of the ease with which a system can be moved form one environment to another
		Portability	IEEE1061-1992 (1992)	An attribute that bears on the ability of software to be transferred from one environment to another
		Portability	McCall et al. (1977)	The effort required to transfer the program from one hardware and/or software system environment to another
Habileté du logiciel à s'exécuter correctement d'une façon consistante		Fiabilité (Reliability)	ISO-9126 (1991)	Ensemble d'attributs portant sur l'aptitude du logiciel à maintenir son niveau de service dans des conditions précises et pendant une période déterminée

		Fiabilité	Robillard (1985)	Capacité qu'a un programme d'exécuter correctement toutes ses structures, pour répondre aux besoins de l'usager et être compris par le concepteur
		Reliability	[A85]	To what degree can the system be expected to perform its function without failure? (Does it work accurately without failure?)
		Reliability	Boehm et al. (1978)	A software product possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily
		Reliability	Bowen et al. (1985)	Extend to which the software will perform without any failures within a specified time period
		Reliability	Deutsch et Willis (1988)	Reliability is the extent to which the CSCI consistently performs the functions specified
		Reliability	[EM87]	Extent to which the software will perform without any failures within a specified time period
		Reliability	[G77]	Reliability is the probability that the system will perform satisfactorily (with no malfunctions) for at least a given time interval, when used under stated conditions
		Reliability	IEEE1061-1992 (1992)	An attribute that bears on the capability of software to maintain its level of performance under stated conditions for a stated period of time
		Reliability	McCall et al. (1977)	The extent to which a program can be expected to perform its intended function with required precision
	Prévention contre les conditions hasardeuses	Safety	Deutsch et Willis (1988)	Safety is the absence of hazardous conditions. Software that can prevent a hazard form occurring is defined to be critical
Possibilité de réutiliser les composantes du logiciel dans d'autres applications		Reusability	[A85]	To what extent can the module or program be used in other applications? (Can it be partially or totally reused in other applications to reduce costs?)
		Reusability	Bowen et al. (1985)	Relative effort to convert a software component for use in another application
		Reusability	Deutsch et Willis (1988)	Reusability is the extent of effort required to convert a portion of this CSCI for use in another application
		Reusability	[EM87]	Relative effort to convert a software component for use in another application

		Reusability	McCall et al. (1977)	The extent to which a program (or parts of a program) can be reused in other applications-related to the packaging and scope of the functions that the program performs
Possibilité de survie du logiciel lorsqu'une partie du système est déficiente		Survivability	Bowen et al. (1985)	Extent to which the software will perform and support critical functions without failures within specified time period when a portion of the system is inoperable
		Survivability	Deutsch et Willis (1988)	Survivability is the extent to which the CSCI *continue to perform its required functions even* when a portion of the system has failed
Possibilité de vérifier les performances du logiciel		Testabilité	Robillard (1985)	Facilité avec laquelle on peut démontrer l'exactitude d'un programme, au moyen d'un test
		Testability	[A85]	How much effort will be required to test the structure and correctness of the code? (Can you test it?)
		Testability	Deutsch et Willis (1988)	Testability is the extent of effort required to verify software operation and performance
		Testability	McCall et al. (1977)	The effort required to test a program to ensure that it performs its intended function
		Verifiability	Bowen et al. (1985)	Relative effort to verify the specified software operation and performance
		Verifiability	[EM87]	Relative effort to verify the specified software operation and performance
Compréhensibilité des composantes du logiciel		Compréhensibilité	Robillard (1985)	Facilité avec laquelle on peut comprendre la fonction d'un programme et la façon de la réaliser en lisant le programme source et sa documentation.
Facilité d'utilisation du logiciel		Commodité	Robillard (1985)	Facilité d'utilisation d'un logiciel
		Facilité d'utilisation (Usability)	ISO-9126 (1991)	Ensemble d'attributs portant sur l'effort nécessaire pour l'utilisation et sur l'évaluation individuelle de cette utilisation par un ensemble défini ou implicite d'utilisateurs
		Usability	[A85]	How much effort will the user expend to learn and use the system input and output? (Can the computer center run it? Can the user operate it easily?)

		Usability (Human engineering)	Boehm et al. (1978)	A software product possesses the characteristic usability to the extent that it is convenient and practicable to use (A software product possesses human engineering to the extent that it fulfills its purpose without wasting user's time and energy or degrading their morale)
		Usability	Bowen et al. (1985)	Relative effort for using software (training and operation) (e.g., familiarization, input preparation, execution, output interpretation)
		Usability	Deutsch et Willis (1988)	Usability is the time and effort required to learn the human interface with the CSCI, prepare input, and interpret the output of the CSCI
		Usability	[EM87]	Relative effort for training or software operation (e.g., familiarization, input preparation, execution, output interpretation)
		Usability	IEEE1061-1992 (1992)	An attribute that bears on the effort needed for use (including preparation for used and evaluation of results) and on the individual assessment of such use by users
		Usability	McCall et al. (1977)	The effort required to learn, operate, prepare input, and interpret output of a program
		Adaptability	Arthur et al. (1993)	

3 .3. Classification des critères de qualité

Tableau 3 .2: classification des critères de qualité de la littérature

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
Caractéristiques des fonctions du logiciel	Présence et adéquation des fonctions	Aptitude (Suitability)	ISO-9126 (1991)	Attributs du logiciel portant sur la présence et l'adéquation d'une série de fonctions pour des tâches données
		Completeness	IEEE1061-1992 (1992)	The degree to which the software possesses necessary and sufficient functions to satisfy user needs
		Correctness	IEEE1061-1992 (1992)	The degree to which all software functions are specified
	Généralité des fonctions	Generality	[A84]	Software attributes that expand the usefulness of the functions beyond the existing module
		Generality	[DW88]	Generality is the software characteristic that ensures the breadth of the functions performed with respect to the application
		Generality	Bowen et al. (1985)	Those characteristics of software which provide breadth to the functions performed with respect to the application
		Generality	McCall et al. (1977)	The breadth of potential application of program components
	Complétude des fonctions	Completeness	[A84]	Software attributes that provide a full implementation of the functions required
		Completeness	[DW88]	Completeness is the software characteristic that ensures full implementation of the function required
		Completeness	Boehm et al. (1978)	A software product possesses the characteristic completeness to the extent that all of its parts are present and each of its parts is fully developed
		Completeness	Bowen et al. (1985)	Those characteristics of software which provide full implementation of the functions required
		Completeness	McCall et al. (1977)	The degree to which full implementation of required function has been achieved
		Intégrité	Robillard (1985)	Caractéristique d'un programme dont toutes les composantes sont présentes, et dont chacune est entièrement développée

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
	Distributivité des fonctions	Distributeness	Bowen et al. (1985)	Those characteristics of software which determine the degree to which software functions are geographically or logically separated within the system
		Distributivity	[DW88]	Distributivity is the degree to which software functions are geographically or logically separated within the system
Implantation et migration du logiciel à un environnement donné	Adaptation à différents environnements	Facilité d'adaptation (Adaptability)	ISO-9126 (1991)	Attributs du logiciel portant sur la possibilité de son adaptation à différents environnements donnés sans que l'on ait recours à d'autres actions ou moyens que ceux prévus à cet effet pour le logiciel considéré
		Reutilisability	IEEE1061-1992 (1992)	The degree to which the software can be reused in applications other than the original application
	Facilité d'installation	Facilité d'installation (Installability)	ISO-9126 (1991)	Attributs du logiciel portant sur l'effort nécessaire pour installer le logiciel dans un environnement donné
		Installability	IEEE1061-1992 (1992)	The effort required to adjust software to a new environment
	Interchangeabilité	Compatibility	IEEE1061-1992 (1992)	The degree to which new software can be installed without changing environments and conditions that were prepared for the replaced software
		Interchangeabilité (Replaceability)	ISO-9126 (1991)	Attributs du logiciel portant sur la possibilité et l'effort pour l'utiliser à la place d'un autre logiciel donné dans le même environnement
Caractéristiques de l'information externe du logiciel	Exactitude des calculs et des sorties	Accuracy	[A84]	Software attributes that provide precision in calculations and outputs
		Accuracy	Deutsch et Willis (1988)	Accuracy is the software characteristic that provides the required precision in calculations and outputs
		Accuracy	Boehm et al. (1978)	A software product possesses accuracy to the extent that its outputs are sufficiently precise to satisfy their intended use
		Accuracy	Bowen et al. (1985)	Those characteristics of software which provide the require precision in calculations and outputs
		Accuracy	McCall et al. (1977)	The precision of computations and control
		Exactitude (Accuracy)	ISO-9126 (1991)	Attributs de logiciel portant sur la fourniture de résultats ou d'effets justes ou convenus
		Exactitude	Robillard (1985)	Précision que doivent avoir les sorties du programme pour correspondre aux résultats attendus

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
Caractéristiques de l'information interne du logiciel	Accessibilité aux composantes	Accessibility	Boehm et al. (1978)	A software product possesses accessibility to the extend that it facilitates the selective use of its components
		Auditability	[A84]	Software attributes that provide for easy auditing of the software, data, and results
		Document accesibility	Bowen et al. (1985)	Those characteristics of software which provides for easy access to software and selective use of its components
		System accessibility	Deutsch et Willis (1988)	System accessibility is the software characteristic that ensures control and auditing of access to the software and data
		System accessibility	Bowen et al. (1985)	Those characteristics of software which provide for control and audit of access to the software and data
	Compréhensi-bilité des composantes internes	Complexity	[A84]	Software attributes that decrease the program's or module's clarity
		Legibility	Boehm et al. (1978)	A software product possesses legibility to the extent that its function and those of its component statements are easily discerned by reading the code
		Lisibilité	Robillard (1985)	Facilité pour l'usager à reconnaître les structures d'un programme, à l'aide de la documentation interne et externe. Un programme lisible peut être lu sans fatigue et sans ennui
		Simplicité	Robillard (1985)	Caractéristique qui permet, après lecture des énoncés d'un programme, d'en comprendre et d'en retracer mentalement les informations, telles la logique du programme, la structure des données, les interactions modulaires
		Simplicity	[A84]	Software attributes that provide and implementation of the functions in the most understandable manner
		Simplicity	Deutsch et Willis (1988)	Simplicity is the software characteristic that ensures definition and implementation of functions in the most direct and understandable way
		Simplicity	Bowen et al. (1985)	Those characteristics of software which provide for definition and implementation of functions in the most noncomplex and understandable manner
		Simplicity	McCall et al. (1977)	The degree to which a program can be understood without difficulty

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		System clarity	Deutsch et Willis (1988)	System clarity is the software characteristic that ensures the clear description of program structure in a direct, understandable manner
		System clarity	Bowen et al. (1985)	Those characteristics of software which provide for clear description of program structure in a non-complex and understandable manner
		Understandability	Boehm et al. (1978)	A software product possesses the characteristic understandability to the extent that the purpose of the product is clear to the evaluator
	Cohérence interne des composantes	Cohérence interne	Robillard (1985)	Uniformité d'un programme dans sa notation, sa terminologie et son symbolisme
		Consistency	[A84]	Software attributes that provide uniform design and implementation techniques and documentation
		Consistency	Deutsch et Willis (1988)	Consistency is the software characteristic that ensures uniform design and implementation techniques and notations
		Consistency	Bowen et al. (1985)	Those characteristics of software which provide for uniform design and implementation techniques and notation
		Consistency	McCall et al. (1977)	The use of uniform design and documentation techniques throughout the software development project
		Internal consistency	Boehm et al. (1978)	A software product possesses the characteristic internal consistency to the extent that it contains uniform notation, terminology, and symbology within itself
	Concision des composantes	Conciseness	Boehm et al. (1978)	A software product possesses the characteristic conciseness to the extent that no excessive information is present
		Conciseness	McCall et al. (1977)	The compactness of the program in terms of lines of code
		Concision	[A84]	Software attributes that implement a function in the minimum amount of code
		Concision	Robillard (1985)	Caractéristique d'un programme ou d'un module ne contenant que les informations nécessaires et suffisantes (absence d'excès d'informations)
	Description des composantes	Self descriptiveness	Boehm et al. (1978)	A software product possesses self-descriptiveness to the extent that is contains enough information for a reader to determine its objectives, assumptions, constraints, inputs, outputs, components, and status
		Self descriptiveness	Bowen et al. (1985)	Those characteristics of software which provide explanation of the implementation of functions

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		Self documentation	McCall et al. (1977)	The degree to which the source code provides meaningful documentation
		Self-descriptiveness	Deutsch et Willis (1988)	Self-descriptiveness is the software characteristic that ensures explanation of the implementation of functions
		Self-descriptiveness	[EM87]	Descriptiveness of implementation programming language
		Self-documentation	[A84]	Software attributes that explain the function of the software
Caractéristiques de l'utilisation du logiciel	Maîtrise de l'exploitation des entrées et des sorties	Communicati-veness	Boehm et al. (1978)	A software product possesses the characteristic communicativeness to the extent that it facilitates the specification of inputs and provides outputs whose form and content are easy to assimilate and useful
		Facilité d'apprentissage (Learnability)	ISO-9126 (1991)	Attributs du logiciel portant sur l'effort que doit faire l'utilisateur pour apprendre son application (par exemple maîtrise de l'exploitation des entrées, des sorties)
		Interactivité	Robillard (1985)	Caractéristique qui permet au programme de faciliter la spécification des entrées et de fournir des sorties dont la forme et le contenu sont utiles et faciles à saisir
	Facilité de compréhen-sion	Ease of learning	IEEE1061-1992 (1992)	The degree to which user effort required to understand software is minimized
		Facilité de compréhension (Understandability)	ISO-9126 (1991)	Attributs du logiciel portant sur l'effort que doit faire l'utilisateur pour reconnaître la logique et sa mise en œuvre
		Understandability	IEEE1061-1992 (1992)	The amount of user effort required to understand the software
	Auto-formation des utilisateurs	Training	[A84]	Software attributes that provide transitions from the current environment to the new system
		Training	Deutsch et Willis (1988)	Training ensures initial familiarization with and subsequent transition from the current operation of the software
		Training	Bowen et al. (1985)	Those characteristics of software which provide transition from current operation and provide initial familiarization
		Training	McCall et al. (1977)	The degree to which the software assists in enabling new users to apply the system
	Facilité d'exploitation	Facilité d'exploitation (Operability)	ISO-9126 (1991)	Attributs du logiciel portant sur l'effort que doit faire l'utilisateur pour exploiter et contrôler son exploitation

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		Operability	[A84]	Software attributes that determine the ease or difficulty of operation of the software
		Operability	Deutsch et Willis (1988)	Operability is the ease by which a person can use the software
		Operability	Bowen et al. (1985)	Those characteristics of software which determine operations and procedures concerned with operation of software ans which provide useful inputs ans outputs which can be assimilated
		Operability	IEEE1061-1992 (1992)	The degree to which the operation of softwares matches the purpose, environment, and physiological characteristics of users, including ergonomic factors such as colors, shape, soun, etc
		Operability	McCall et al. (1977)	The ease of operation of a program
	En accord avec les caractéristiques psychologiques des utilisateurs	Communicativeness	IEEE1061-1992 (1992)	The degree to which the software is designed in accordance with the psychological characteristics of users
Comportement du logiciel vis-à-vis des anomalies	Continuité d'opération et possibilité de récupération sous des conditions non nominales	Anomaly management	Deutsch et Willis (1988)	Anomaly management is the software characteristic that ensures continuity of operations under and recovery from abnormal conditions
		Anomaly management	Bowen et al. (1985)	Those characteristics of software which provide for continuity of operations under and recovery from non-nominal conditions
		Error tolerance	IEEE1061-1992 (1992)	The degree to which software will continue to work without a system failure that will cause damage to the users. Also, the degree to which software includes degraded operation and recovery functions
		Availability	IEEE1061-1992 (1992)	The degree to which software remains operable in the presence of systems failures
		Error tolerance	[A84]	Software attributes continuity of operation under adverse conditions
		Reconfigurability	Deutsch et Willis (1988)	Reconfigurability is the software characteristic that ensures continuity of system operation when one or more processor, storage units, or communication links fail

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		Reconfigurability	Bowen et al. (1985)	Those characteristics of software which provide for continuity of system operation when one or more processors, storage units, or communication links fails
		Résistance	Robillard (1985)	Caractéristique qui permet au programme de pouvoir continuer son traitement malgré quelques violations des hypothèses de ses spécifications
		Tolérance aux fautes (Fault tolerance)	ISO-9126 (1991)	Attributs du logiciel portant sur son aptitude à maintenir un niveau de service donné en cas de défaut du logiciel ou de violation de son interface
		Possibilité de récupération (Recoverability)	ISO-9126 (1991)	Attributs du logiciel portant sur ses capacités de rétablir son niveau de service et de restaurer les informations directement affectées en cas de défaillance, et sur le temps et l'effort nécessaires pour le faire
	Prévention des anomalies	Safety management	Deutsch et Willis (1988)	Safety management is the software characteristic that separates critical and non-critical software to prevent unsafe conditions from occurring
	Fréquence et sévérité des défaillances	Error tolerance	McCall et al. (1977)	The damage that occurs when the program encounters an error
		Frequency/severity of failure	Grady et Caswell (1987)	
		Maturité (Maturity)	ISO-9126 (1991)	Attributs du logiciel portant sur la fréquence des défaillances dues à des défauts du logiciel
		Mean time to failure	Grady et Caswell (1987)	
	Instrumentation du logiciel	Instrumentation	[A84]	Software attributes that provide measurements of software usage or the identification of errors
		Instrumentation	McCall et al. (1977)	The degree to which the program monitors its own operation and identifies errors that do occur
	Absence d'erreurs	Error free	[RPRB94]	
		Nondeficiency	IEEE1061-1992 (1992)	The degree to which software does not contain undetected errors
Comportement du logiciel vis-à-vis des reccources et du temps	Comportement vis-à-vis des ressources	Comportement vis-à-vis des ressources (Ressource behavior)	ISO-9126 (1991)	Attributs du logiciel portant sur la quantité de ressources utilisées et sur la durée de leur utilisation lorsqu'il exécute sa fonction

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		Efficience	Robillard (1985)	Capacité d'exécution des structures qui composent un programme, sans gaspillage des ressources de la machine (mémoire centrale, mémoire secondaire, canaux de communication, temps d'exécution, etc.)
		Resource economy	IEEE1061-1992 (1992)	Capability of software to perform specified functions under stated or implied conditions, using appropriate amounts of ressources
		Effectiveness communication	Bowen et al. (1985)	Those characteristics of the software which provide for minimum utilization of communications resources in performing functions
		Efficiency of communication	Deutsch et Willis (1988)	Efficiency of communication is the software characteristic that ensures minimum utilization of communication ressources in performing functions
		Effectiveness processing	Bowen et al. (1985)	Those characteristics of the software which provide for minimum utilization of processing resources in performing functions
		Efficiency of processing	Deutsch et Willis (1988)	Efficiency of processing is the software characteristic that ensures minimum utilization of processing ressources in performing functions
		Effectiveness storage	Bowen et al. (1985)	Those characteristics of the software which provide for minimum utilization of storage resources
		Efficiency of storage	Deutsch et Willis (1988)	Efficiency of storage is the software characteristic that ensures minimum utilization of storage ressources
	Comportement vis-à-vis du temps	Comportement vis-à-vis du temps (Time behavior)	ISO-9126 (1991)	Attributs du logiciel portant sur les temps de réponse et de traitement ainsi que sur les débits lors de l'exécution de sa fonction
		Execution efficiency	[A84]	Software attributes that minimize processing time
		Execution efficiency	McCall et al. (1977)	The run-time performance of a program
		Time economy	IEEE1061-1992 (1992)	Capability of software to perform specified functions under stated or implied conditions within appropriate time frames
Evolution et correction du logiciel		Facilité de modification (Changeability)	ISO-9126 (1991)	Attributs du logiciel portant sur l'effort nécessaire pour modifier, remédier aux défauts ou changer d'environnement
	Evolution	Augmentability	Deutsch et Willis (1988)	Augmentability is the software characteristic that ensures expansion of capability for functions and data

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		Augmentability	Boehm et al. (1978)	A software product possesses augmentability to the extent that it easily accommodates expansions in data storage requirements or component computational functions
		Augmentability	Bowen et al. (1985)	Those characteristics of software which provide for expansion of capability for functions and data
		Expandability	[A84]	Software attributes that provide for expansion of the data or program functions
		Expandability	IEEE1061-1992 (1992)	The degree of effort required to improve or modify the efficiency or functions of software
		Expandability	McCall et al. (1977)	The degree to which architectural, data, or procedural design can be extended
		Modifiability	Boehm et al. (1978)	A software product possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined
	Facilité de correction	Correctability	IEEE1061-1992 (1992)	The degree of effort required to correct errors in software and cope with user complaints
	Facilité d'analyse	Facilité d'analyse (Analyzability)	ISO-9126 (1991)	Attributs du logiciel portant sur l'effort nécessaire pour diagnostiquer les déficiences ou les causes de défaillance, ou pour identifier les parties à modifier
	Risque et effets des modifications	Stabilité (Stability)	ISO-9126 (1991)	Attributs du logiciel portant sur le risque des effets inattendus des modifications
Vérifiabilité du logiciel	Test	Facilité de test (Testability)	ISO-9126 (1991)	Attributs du logiciel portant sur l'effort nécessaire pour valider le logiciel modifié
		Testability	Boehm et al. (1978)	A software product possesses the characteristic testability to the extent that it facilitates the establishment of acceptance criteria and supports evaluation of its performance
		Testability	IEEE1061-1992 (1992)	The effort required to test software
	Caractéristiques permettant de suivre le développement	Visibility	Bowen et al. (1985)	Those characteristics of software which provide status monitoring of the development and operation
		Visibility	Deutsch et Willis (1988)	Visibility is the software characteristic that provides monitoring of the development and operation of the software

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
Indépendance du logiciel à des considérations externes	*Indépendance vis-à-vis les systèmes logiciels et le matériel*	Application independence	Bowen et al. (1985)	Those characteristics of software which determine its nondependency on database system, microcode, computer architecture, and algorithms
		Application independence	Deutsch et Willis (1988)	Application independence is the software characteristic that ensures that the software is not dependent on any data-base system, microcode, computer architecture, or algorithms
		Independence	Deutsch et Willis (1988)	Independence is the software characteristic that ensures that it does not depend on its environment (the computing system, operating system, utilities, I/O routines, and libraries)
		Independence	[EM87]	Software system independence, machine independence and degree of independence
		Independence	Bowen et al. (1985)	Those characteristics of software which determine its non-dependency on software environment (computing system, operating system, utilities, input, output routines, libraries)
		Device-independence	Boehm et al. (1978)	A software product possesses the characteristic device-independence to the extent that it can be executed on computer hardware configurations other than its current one
		Hardware independence	[A84]	Software attributes that determine its dependence on the hardware
		Hardware independence	IEEE1061-1992 (1992)	The degree to which software does not depend on specific hardware environments
		Hardware independence	McCall et al. (1977)	The degree to which the software is decoupled from the hardware on which it operates
		Universalité	Robillard (1985)	Transparence d'un programme à son environnement matériel: en d'autres termes, l'universalité suppose que le programme soit le plus indépendant possible des environnements matériels où il est susceptible d'être exécuté
		Autonomy	Bowen et al. (1985)	Those characteristics of software which determine its non-dependency on interfaces and functions
		Software independence	IEEE1061-1992 (1992)	The degree to which software does not depend on specific software environments
		Software system independence	[A84]	Software attributes that determine its dependence on the software environment - extensions of the language, operating system, data base management system, and so on

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		Software system independence	McCall et al. (1977)	The degree to which the program is independent of nonstandard programming language features, operating system characteristics, and other environmental constraints
Contrôle et protection des composantes du logiciel à des usages non autorisés		Sécurité (Security)	ISO-9126 (1991)	Attributs du logiciel portant sur son aptitude à empêcher tout accès non autorisé (accidentel ou délibéré) aux programmes et données
		Security	[A84]	Software attributes that provide for control and protection of the software and data
		Security	IEEE1061-1992 (1992)	The degree to which software can detect and prevent information leak, information loss, illegal use, and system ressource destruction
		Security	McCall et al. (1977)	The availability of mechanisms that control or protect programs and data
Interaction du logiciel avec d'autres systèmes		Interopérabilité (Interoperability)	ISO-9126 (1991)	Attributs de logiciel portant sur sa capacité à interagir avec des systèmes donnés
		Interoperability	IEEE1061-1992 (1992)	The degree to which software can be connected easily with other systems and operated
	Compatibilité entre des systèmes	System compatibility	Bowen et al. (1985)	Those characteristics of software which provide the hardware, software, and communication compatibility of two systems
		System compatibility	Deutsch et Willis (1988)	System compatibility is the software characteristic that ensures the hardware, software, and communication compatibility of two CSCIs
Standardisation des composantes du logiciel		Commonality	Bowen et al. (1985)	Those characteristics of software which provide for the use of interface standards for protocols, routines, and data representations
		Commonality	Deutsch et Willis (1988)	Commonality is the software characteristic that ensures the use of interface standards for protocols, routines, and data representations
	Standardisation des données	Data commonality	[A84]	Software attributes that provide standard data representations
		Data commonality	McCall et al. (1977)	The use of standard data structures and types throughout the program

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
	Standardisation des communications entre composantes	Communication commonality	[A84]	Software attributes that use standard protocols and interface routines
		Communication commonality	McCall et al. (1977)	The degree to which standard interfaces, protocols, and bandwidths are used
	Standardisation des fonctions entre les systèmes	Functional overlap	Bowen et al. (1985)	Those characteristics of software which provide common functions to both systems
		Functional overlap	Deutsch et Willis (1988)	Functional overlap is the commonality of functions between CSCI's
	Standardisation des fonctions à l'intérieur d'un système	Functionnal scope	Bowen et al. (1985)	Those characteristics of software which provide commonality of functions among applications
		Functional scope	Deutsch et Willis (1988)	Functional scope is the commonality of function within a CSCI
	Conformité aux standards	Auditability	McCall et al. (1977)	The ease with which conformance to standards can be checked
		Conformité réglementaire (Compliance)	ISO-9126 (1991)	Attributs du logiciel selon lesquels il respecte l'application des normes, des conventions, des réglementations de droit ou des prescriptions similaires
		Conformité relative aux règles de portabilité (Conformance)	ISO-9126 (1991)	Attributs du logiciel permettant à celui-ci de se conformer aux normes ou conventions ayant trait à la portabilité
Structure interne du logiciel		Structuredness	Boehm et al. (1978)	A software product possesses the characteristic structureness to the extent that it possesses a definite pattern of organization of its interdependent parts
	Modularité	Modularité	Robillard (1985)	Pertinence de la fonction de chaque module ainsi que ses interactions avec les autres modules
		Modularity	[A84]	Software attributes that provide a structure of highly indepence modules
		Modularity	Deutsch et Willis (1988)	Modularity is the software characteristic that ensures a highly cohesive component structure with optimum coupling

Concepts	Sous-concepts	Appelation d'origine	Auteurs	Descriptions
		Modularity	Bowen et al. (1985)	Those characteristics of software provide a structure of highly cohesive components with optimum coupling
		Modularity	McCall et al. (1977)	The functional independence of program component
Liens entre les caractéristiques internes et externes du logiciel		Cohérence externe	Robillard (1985)	Correspondance entre le contenu d'un programme d'une part, et les spécifications et la documentation d'autre part
		External consistency	Boehm et al. (1978)	The product possesses the characteristic external *consistency to the extent that the content is* traceable to the requirements
		Traceability	[A84]	Software attributes that provide for a link from requirements to the implemented software
		Traceability	Deutsch et Willis (1988)	Traceability is the software characteristic that provides a thread of origin from the implementation to the requirements with respect to the specified development envelope and operational environment
		Traceability	Bowen et al. (1985)	Those characteristics of software which provide a thread of origin from the implementation of the requirements with respect to the specified development envelope and operational environment
		Traceability	McCall et al. (1977)	The ability to trace a design representation or actual program component back to requirements
Divers		Virtuality	Bowen et al. (1985)	Those characteristics of software which present a system that does not require user knowledge of the physical, logical or topological characteristics

ANNEXE IV
MÉTHODES POUR ÉVALUER LA PERTINENCE DES MÉTRIQUES LOGICIELLES

4.1. Introduction

Dans cette annexe, nous présentons plusieurs méthodes pour évaluer la pertinence des métriques logicielles. Ces méthodes, issues des travaux de IEEE 1061-1992 (1992) et de Boehm et al. (1978), s'avèrent particulièrement utiles quand vient le temps de choisir des métriques logicielles pour peupler les modèles de qualité.

4.2. Critères pour évaluer l'utilité des métriques

Plusieurs critères furent considérés pour évaluer l'utilité des métriques dans l'étude de Boehm et al. (1978). Bien que ces critères visent les métriques de produits associées au code, ils peuvent être adaptés à n'importe quelle métrique de produits ou processus. Ces critères sont décrits dans les paragraphes suivants.

Corrélation avec la qualité logicielle

Pour chaque métrique supposée mesurer un certain attribut externe, est-ce qu'elle corrèle avec la notion de la qualité logicielle ? Par corrélation positive, les auteurs entendent que la plupart des programmes avec un haut score pour une métrique posséderaient également l'attribut externe associé. Évidemment, une définition statistique plus précise pourrait être énoncée, mais l'évaluation est plutôt subjective à ce point.

L'échelle suivante fut utilisée pour classer chaque métrique :

- A- corrélation positive très élevée ; presque tous les programmes avec un haut score pour la métrique posséderont l'attribut externe associé ;
- AA- corrélation positive élevée ; une bonne majorité (disons 75-90%) des programmes avec un haut score pour la métrique posséderont l'attribut externe associé ;
- U- typiquement (disons 50-75%) des programmes avec un haut score pour la métrique posséderont l'attribut externe associé ;
- S- certains programmes avec un haut score pour la métrique posséderont l'attribut externe associé.

Bénéfices potentiels des métriques

Certaines métriques fournissent des données aidant la prise de décision et des indices qui sont très importants à la fois pour le développeur et l'usager potentiel d'un produit logiciel ; d'autres fournissent de l'information qui est intéressante, possiblement indicatrice de problèmes potentiels, mais qui n'est pas d'une grande perte si la métrique n'a pas un haut

score même si elle est fortement corrélée avec sa caractéristique de qualité associée. Le jugement du bénéfice potentiel dépend évidemment de l'utilisation pour laquelle l'évaluateur évalue le produit.

L'échelle suivante de bénéfices potentiels a été définie :

5- il est extrêmement important que la métrique ait un haut score ; problème potentiel majeur autrement ;
4- il est important que la métrique ait un haut score ;
3- il est assez important que la métrique ait un haut score ;
2- il est relativement important que la métrique ait un haut score ;
1- il est plus ou moins important que la métrique ait un haut score ; pas vraiment de perte autrement.

Quantifiabilité des métriques et faisabilité d'une évaluation automatisée

Bien qu'une métrique peut corréler fortement avec la qualité tout en offrant beaucoup de bénéfices potentiels, il peut être très long ou coûteux de déterminer sa valeur numérique. En fait, si l'évaluation d'une métrique requiert qu'un expert lise un programme et fasse un jugement, la valeur numérique fournira probablement moins d'indices que la compréhension que l'expert gagnera dans le processus d'évaluation. De plus, les métriques nécessitant des évaluateurs experts sont très coûteuses à utiliser. Conséquemment, il est préférable pour les longs programmes d'avoir un algorithme automatisé qui examine le programme et produit des valeurs de métriques. Un outil intermédiaire, qui est généralement plus faisable, est un vérificateur automatisé de conformité auquel l'utilisateur doit fournir une liste de caractéristiques de qualité désirées.

Dans l'évaluation, un jugement peut être fait pour déterminer quelle combinaison de méthodes de quantification fournirait le classement le plus efficace en regard au coût pour la métrique, en utilisant l'ensemble suivant d'options :

AL- peut être fait efficacement par l'entremise d'un algorithme automatisé ;
CC- peut être fait efficacement par l'entremise d'un vérificateur automatisé de conformité ;

UR- nécessite un lecteur ou inspecteur non formé ;
TR- nécessite un lecteur ou inspecteur formé ;
ER- nécessite un lecteur ou inspecteur expert ;
EX- nécessite que le programme soit exécuté.

Automatiser certaines évaluations, telles le comptage de la longueur moyenne des modules ou vérifier pour la présence de certaines sortes de matériel auto-descriptif peut être réalisé d'une façon assez directe. Automatiser d'autres évaluations, telles balayer globalement le programme pour des sous-expressions répétées et garantir que les composantes dans les sous-expressions ne sont pas modifiées entre les répétitions, est possible mais plus difficile. D'autres, telles juger la description du matériel auto-descriptif, sont presque impossibles à automatiser. Ainsi, certains outils automatisés peuvent fournir un support utile mais partiel pour l'évaluation de la qualité, laissant le reste à être complété par un lecteur humain.

Les échelles suivantes ont été utilisées pour classer chacune des métriques selon la facilité et la complétude d'une évaluation automatisée :

facilité de développer une évaluation automatisée

E- facile de développer un algorithme automatisé ou un vérificateur de conformité ;
M- modérément difficile de développer un algorithme automatisé ou un vérificateur de conformité ;
D- difficile de développer un algorithme automatisé ou un vérificateur de conformité.

complétude de l'évaluation automatisée

C- l'algorithme ou le vérificateur fournit un évaluation totale de la métrique ;
P- l'algorithme ou le vérificateur fournit une évaluation partielle de la métrique ;
I- l'algorithme ou le vérificateur fournit des résultats peu concluants.

Tous les critères précédant furent appliqués aux métriques développées dans l'étude de Boehm (1978). Certains exemples sont donnés dans le Tableau 4 .1.

Tableau 4 .1 : exemple d'évaluation des métriques

Indépendance aux matériels DI-1	Est-ce que les calculs sont indépendants de la longueur des mots de l'ordinateur pour atteindre la précision requise des schémas de stockage ?	A	5	AL+EX	E	P
DI-4	Est-ce que les énoncés dépendant du matériel ont été notés et commentés ?	A	5	AL	M	P
Complétude CP-1	Est-ce que le programme contient un utilitaire pour initialiser la mémoire centrale avant son utilisation ?	A	5	AL	E	P
CP-2	Est-ce que le programme contient un utilitaire pour le positionnement correct des périphériques d'entrées/sorties avant son utilisation ?	A	5	CC	E	P
Précision AR-1	Est-ce que les méthodes numériques utilisées par le programme sont consistantes avec les exigences de l'application ?	A	5	TR		
Consistance CS-1	Est-ce que toutes les spécifications des ensembles de variables globales (i.e., ceux apparaissant dans deux sous-programmes ou plus) sont identifiées ?	AA	4	AI	E	C
CS-2	Est-ce que le type d'une variable est consistant pour toutes les utilisations ?	A	5	AL	E	P

Une explication du Tableau 4 .1 devrait éclaircir la démarche d'évaluation. La métrique (DI-1) fut trouvée corrélant hautement avec l'indépendance vis-à-vis le matériel (classement "A") et étant extrêmement importante pour l'indépendance vis-à-vis le matériel (classement "5"). Il fut trouvé qu'une combinaison d'un algorithme automatisé d'exécution et un lecteur formé serait généralement plus efficace pour déterminer le degré selon lequel un produit logiciel possède la caractéristique (classement "AL+EX+TR"). Un algorithme automatisé pourrait vérifier le format des énoncés pour l'indépendance aux matériels et relever les constantes extra-précises. Ces vérifications seraient facilement automatisables (classement "E"), mais fourniraient seulement des résultats partiels (classement "P").

4 .3. Analyse coût-bénéfice de l'application des métriques

Selon Boehm et al. (1978), l'application des métriques au processus de développement de projets logiciels d'envergure devrait donner des bénéfices à long terme et aussi des bénéfices immédiats. Les bénéfices à long terme proviennent des projets de développement futurs, i.e. autres que celui courant. Les bénéfices immédiats sont liés aux coûts qui peuvent être sauvés durant la mise au point du logiciel, lors des tests durant le développement et lors de l'opération et de la maintenance du produit. Les sections 4 .3.1 et 4 .3.2 discutent plus en détails des bénéfices à long et à court terme respectivement. La section 4 .3.3 présente un modèle d'analyse des économies résultant de l'application des métriques.

4 .3.1. Bénéfices à long terme

L'intérêt principal de l'application des métriques pour évaluer certains attributs de projets logiciels d'envergure est l'utilité potentielle des données cueillies sur chaque logiciel. La collecte et la catégorisation des valeurs de métriques permettent d'extrapoler et de prédire les propriétés de logiciels futurs. En prenant les métriques comme un outil pour réduire le coût de rencontrer les exigences (les bénéfices immédiats), l'application consistante de métriques à plusieurs logiciels devrait amener à l'amélioration de cet outil.

4 .3.2. Bénéfices immédiats

Il est postulé que l'application de métriques, particulièrement durant les premières phases d'un projet logiciel, mais aussi durant les phases d'opération et de maintenance, réduira significativement les erreurs logicielles. Ces économies postulées sont obtenues en payant le prix pour appliquer les métriques. Conséquemment, la relation entre le coût total d'un projet et le coût de l'application d'un ensemble approprié de métriques doit être analysé. Il est connu que, pour trouver l'ensemble approprié de métriques, il faut recourir à plusieurs projets logiciels pour trouver la pertinence des métriques. Or, nous prenons pour acquis dans cette discussion qu'un ensemble adéquat de métriques a déjà été défini. Le coût d'appliquer une métrique consiste aux items suivants :

- travail pour l'analyse et l'évaluation numérique ;
- coûts en ressources matérielles pour l'automatisation des métriques, incluant l'ajout de temps système (utilisation de la mémoire et du CPU) dû à l'utilisation de code instrumenté durant la mise au point et les phases de tests ;
- travail des gestionnaires et des analystes/programmeurs dans la situation où une valeur basse d'une métrique résulte en une décisions de faire des changements au design, au code et à la documentation ;
- délais des échéanciers qui résultent en la perte de primes de rendement en raison de l'application des métriques ;
- la gestion de la discipline comme une fonction de projets devrait également être inclue dans le coût d'application des métriques. Ces coûts incluent : la préparation des procédures du projet, les sessions d'endoctrinement, la préparation de rapports, l'entoilage avec les superviseurs et le client et l'évaluation de l'impact du projet.

L'évaluation des coûts économisés doit considérer à la fois les erreurs qui ne surviennent pas parce que les métriques ont été appliquées et aussi les erreurs qui surviennent et qui auraient été évitées par l'application des métriques.

Une façon d'évaluer la réduction des coûts liés aux erreurs par l'application des métriques est de sélectionner deux routines ou plus de difficulté et de grosseur estimées équivalentes. Durant le développement, il faut appliquer les métriques appropriées à certaines routines et ne pas les appliquer aux autres. Si possible, il faut éviter les différences de compétence et d'expérience des programmeurs et de tous autres facteurs qui pourraient fausser les résultats. Il est nécessaire que la cueillette des coûts soit plus détaillée qu'à l'habitude. Les heures de travail devront dont être comptabilisées par type d'activités. Une autre façon d'évaluer la réduction possible des coûts résultant de l'application des métriques est de développer deux routines ou plus par deux équipes à partir des mêmes spécifications. Les routines seront développées selon différents critères, i.e., nombre minimal d'instructions pour un programme et aucune restriction pour l'autre, etc.

4.3.3. Modèle d'analyse des économies résultant de l'application des métriques

Un modèle simple pour analyser l'effet d'appliquer des métriques durant les phases de développement d'un projet logiciel est présentée dans cette sous-section. Pour faire cette analyse, une approche paramétrique est utilisée pour définir le coût des erreurs en fonction de la phase de développement et le coût d'application des métriques.

Le modèle prend pour acquis que le développement logiciel consiste aux huit phases suivantes :

1. définition des exigences ;
2. design ;
3. codage ;
4. mise au point et tests lors du développement ;
5. validation ;
6. tests d'acceptance ;
7. tests d'intégration ;
8. livraison - opération et maintenance.

Le coût encouru pour trouver toutes les erreurs d'un certain type pour chacune des phases ci-haut est noté par :

C_i, i=1,2,...,8 où généralement $C_1<C_2<...<C_8$

Supposons que les erreurs de type j arrivent durant la phase i_j lorsque les métriques ne sont pas utilisées. L'utilisation des métriques devrait permettre de détecter les erreurs de type j durant la phase k_j. Le coût d'application des métriques jusqu'à la phase k_j est $\sum_{i=1}^{k_j} m_i = M_{k_j}$ où m_i est le coût d'appliquer les métriques à la $i^{ième}$ phase pour chaque type d'erreur.

Les économies dues à l'application des métriques sont $C_{ij} - C_{kj}$ pour les erreurs de type j. Ces économies sont réduites de M_{kj}. Conséquemment, les économies nettes pour les erreurs de type j sont $K_j = C_{ij} - C_{kj} - M_{kj}$ et les économies totales sont $K_T = \sum_{j=1}^{N} K_j$ où N est le nombre total de types d'erreurs différents.

Le ratio $C_R = \frac{C_U - C_K - M_K}{C_U}$ des économies dues à l'utilisation des métriques sur le coût des erreurs n'ayant pas utilisé les métriques est probablement plus adéquat puisque c'est une mesure sans dimension de la performance des métriques. Évidemment $C_R \leq 1.0$ et peut être négatif. Plus C_R est élevé, plus la performance est supérieure. La quantité $\overline{C_T} = \frac{1}{N}\sum_{j=1}^{N} C_{Rj}$ est le ratio moyen d'économies.

Pour simplifier, nous prenons pour acquis que le coût des erreurs de chaque type augmente linéairement en fonction de la phase logicielle, soit $C_i = C_o i$. Deuxièmement, il est pris pour acquis que le coût d'application des métriques est indépendant de la phase, ainsi,

$$m_i = m, \quad i = 1,2,...,8$$

et conséquemment

$$M_{kj} = mk_j$$

Donc,

$$\begin{aligned} K_j &= C_o(i_j - k_j) - mk_j \\ &= C_o i_j - k_j(C_o + m) \end{aligned}$$

conséquemment

$$\begin{aligned} C_{Rj} &= \frac{C_o i_j - k_j(C_o + m)}{C_o i_j} \\ &= 1 - (\frac{C_o + m}{C_o})(\frac{k_j}{i_j}) \end{aligned}$$

et finalement

$$\begin{aligned} \overline{C_T} &= 1 - \frac{1}{N}(1 + \frac{m}{C_o})\sum_{j=1}^{N}\frac{k_j}{i_j} \\ &= 1 - \frac{1}{N}(1+r)\sum\frac{k_j}{i_j} \qquad ou \quad r = \frac{m}{C_o} \end{aligned}$$

4 .4. Validation des métriques de qualité logicielle

Pour obtenir des informations sur les techniques statistiques utilisées, voir Conover (1971), Gibbons (1971), Kleinbaum et Kupper (1978), ou des références similaires.

4 .4.1. Objectif de la validation des métriques

L'objectif de la validation des métriques est de s'assurer qu'elles peuvent évaluer adéquatement les valeurs des attributs externes.

Jusqu'à présent, selon IEEE std. 1061-1992 (1992), les métriques qui sont utilisées sont rarement validées (c'est-à-dire, il n'est pas démontré par des analyses statistiques que les métriques mesurent les attributs qu'elles sont sensées mesurer). Cependant, il est important que les métriques soient validées avant qu'elles ne soient utilisées pour évaluer la qualité logicielle. Sinon, les métriques peuvent être mal employées (c'est-à-dire, des métriques présentant peu ou pas de liens avec les attributs externes de qualité peuvent être utilisées).

Les attributs externes de qualité peuvent être affectés par une multitude de variables. Une seule métrique, par conséquent, peut ne pas représenter suffisamment un attribut si elle ignore ces autres variables.

4 .4.2. Critères de validité

Pour être considérée valide, une métrique doit démontrer un haut degré d'association avec les attributs qu'elle représente. Ceci est équivalent à représenter fidèlement les conditions de qualité d'un produit ou d'un processus. Une métrique peut être valide selon certains critères de validité et invalide selon d'autres critères.

Les valeurs seuils des éléments ci-dessous doivent être choisies en vue de la validation.

V - le carré du coefficient de corrélation linéaire (« square of the linear correlation coefficient »)

B - coefficient de corrélation d'ordonnancement (« rank correlation coefficient »)

A - erreur de prédiction

P - taux de succès

Un court exemple numérique suit la définition de chaque critère de validité.

A) *Corrélation.* La variation des valeurs d'un attribut externe expliqué par la variation des valeurs de la métrique, qui est donnée par le carré du coefficient de corrélation linéaire (R) entre la métrique et l'attribut correspondant, devrait excéder V, où $R^2>V$. Ce critère évalue s'il y a une association linéaire suffisamment forte entre un attribut et une métrique.

Par exemple, le coefficient de corrélation entre une métrique de complexité et l'attribut fiabilité peut être de 0,8. Le carré donne 0,64. Seulement 64% de la variation de l'attribut est expliqué par la variation de la métrique. Si V a été établi à 0,7, la conclusion serait que la métrique est invalide (c'est-à-dire que la corrélation ou l'association entre la métrique et la fiabilité est insuffisante). Si la relation est démontrée à partir d'un échantillon représentatif de composantes logicielles, la conclusion serait que la métrique est invalide.

B) *Suivi* « Tracking ». Si une métrique M est directement reliée à un attribut externe de qualité F, pour un produit ou processus donné, alors une modification de la valeur de l'attribut de F_{T1} à F_{T2}, au temps T1 et T2, devrait être suivie par une modification de la valeur de la métrique de M_{T1} à M_{T2}, qui s'avère être la même direction (par exemple, si F augmente, M augmente). Si M est relié inversement à F, alors une modification de F devrait être suivie par une modification de M dans la direction opposée (par exemple, si F augmente, M diminue). Pour faire ce test, il faut calculer le coefficient de corrélation d'ordonnancement (« rank correlation coefficient ») (r) à partir de n paires de valeurs de l'attribut et de la métrique. Chacune des paires attribut/métrique est mesurée dans le même point dans le temps, et les n paires de valeurs sont mesurées à n points dans le temps. La valeur absolue de (r) devrait excéder B. Ce critère évalue si une métrique est capable de suivre les modifications dans la qualité du produit ou du processus durant le cycle de vie.

Par exemple, si une métrique de complexité prétend mesurer la fiabilité, alors il est raisonnable de s'attendre à ce qu'un changement de fiabilité d'une composante logicielle soit suivi par un changement approprié de la valeur de la métrique (dans ces circonstances, si la fiabilité du produit augmente, la valeur de la métrique devrait changer dans la direction qui indique que le produit s'améliore). C'est-à-dire, si le temps entre deux défaillances (MTTF) est utilisé pour mesurer la fiabilité et que ce temps équivaut à 1000 heures durant les tests (T1) et 1500 heures durant l'opération (T2), une métrique de complexité dont la valeur est 8 à T1 et 6 à T2, où 6 est moins complexe que 8, suit la fiabilité pour cette composante logicielle. Si cette relation est démontrée sur un échantillon représentatif de composantes logicielles, la conclusion serait que la métrique peut suivre la fiabilité durant le cycle de vie logiciel.

C) *Consistance.* Si les valeurs F1, F2, F*an* d'un attribut externe, correspondant à des produits ou processus 1, 2, *n*, ont la relation F1 > F2 >..., > F*an*, alors les valeurs de la métrique correspondante devraient avoir la relation M1 > M2..., > M*n*. Pour faire ce test, il faut calculer le coefficient de corrélation d'ordonnancement (« ranch corrélation coefficient ») (*r*) entre paires de valeurs (des mêmes composantes logicielles) de l'attribut et de la métrique ; |*r*| devrait excéder *B*. Ce critère évalue s'il y a consistance entre l'ordonnancement des valeurs des attributs d'un ensemble de composantes logicielles et l'ordonnancement des valeurs des métriques pour le même ensemble de composantes logicielles. Ainsi, ce critère est utilisé pour déterminer si une métrique peut ordonnancer d'une façon précise, par rapport à la qualité, un ensemble de produits ou de processus.

Par exemple, si la fiabilité des composantes logicielles X, Y, Z, mesurée par MTTF, est 1000, 1500 et 800 heures respectivement et que les valeurs correspondantes de la métrique de complexité sont 5, 3, 7, où une valeur basse de la métrique est mieux qu'une valeur haute, l'ordonnancement pour la fiabilité et les valeurs de la métrique, avec 1 représentant le rang le plus élevé, sont :

Y	1	1
X	2	2
Z	3	3

Si la relation est démontrée sur un échantillon représentatif de composantes logicielles, la conclusion serait que la métrique est consistante et peut être utilisée pour ordonnancer la qualité des composantes logicielles.

D) *Prévisibilité.* Si une métrique est utilisée au temps T1 pour prédire un attribut externe de qualité pour un produit ou un processus, elle doit prédire un attribut relié Fp_{T2} avec une précision de

$$\left| \frac{Fa_{T2} - Fp_{T2}}{Fa_{T2}} \right| < A$$

où Fa_{T2} est la valeur actuelle de F au temps T2. Ce critère évalue si une métrique est capable de prédire une valeur d'un attribut avec la précision requise.

Par exemple, si une métrique de complexité est utilisée durant le développement pour prédire la fiabilité d'une composante logicielle durant l'opération (T2) comme étant 1200 heures MTTF (Fp_{T2}) et que le MTTF actuel qui est mesuré durant l'opération est 1000 heures (Fa_{T2}), alors l'erreur de prédiction est 200 heures, ou 20%. Si l'erreur de prédiction acceptable (A) est 25%, la précision de la prédiction est acceptable. Si l'habileté de prédire est démontrée sur un échantillon représentatif de composantes logicielles, la conclusion serait que la métrique peut être utilisée pour prédire la fiabilité.

E) « Discriminative Power ». Une métrique devrait être capable de distinguer entre des composantes logicielles de haute qualité (par exemple, MTTF élevé) et des composantes logicielles de basse qualité (par exemple, MTTF bas). Dans ces circonstances, l'ensemble des valeurs de la métrique associées avec les premières composantes devrait être significativement supérieur (ou inférieur) à l'ensemble des valeurs des autres composantes.

Ce critère évalue si une métrique est capable de séparer un ensemble de composantes logicielles de haute qualité d'un ensemble de composantes logicielles de basse qualité. Cette capacité identifie les valeurs critiques pour les métriques pouvant être utilisées pour identifier les composantes logicielles qui peuvent être d'un niveau de qualité inacceptable. Le test Mann-Whitney et le test Chi-carrée pour les différences dans les probabilités (tables de contingence) peuvent être utilisés pour ce test de validation.

Par exemple, si toutes les composantes logicielles avec une valeur d'une métrique de complexité plus grande que 10 (valeur critique) ont une MTTF de 1000 heures et toutes les composantes avec une valeur de la métrique de complexité égale ou inférieure à 10 ont une MTTF de 2000 heures, et que cette différence est suffisante pour passer les tests statistiques, alors la métrique sépare les composantes logicielles de basse et de haute qualité. Si l'habileté de distinguer est démontrée sur un échantillon représentatif de composantes logicielles, la conclusion serait que la métrique peut distinguer entre des composantes de fiabilité élevée et basse.

F) *Fiabilité.* Une métrique devrait démontrer les propriétés de corrélation, de suivi, de consistance, de prévisibilité et de « discriminative power » pour un pourcentage P des applications de la métrique. Ce critère est utilisé pour s'assurer qu'une métrique a passé un test de validité sur un nombre suffisant ou pourcentage d'applications de telle sorte qu'il y aura un niveau de confiance élevé que la métrique peut remplir ses fonctions.

Par exemple, si le taux de succès (P) requis pour valider une métrique de complexité envers le critère de prévisibilité a été établi à 80%, et qu'il y a 100 composantes logicielles, la métrique devrait prédire l'attribut associé avec la précision requise pour au moins 80 composantes.

4 .4.3. Procédure de validation

Avant qu'une métrique soit utilisée pour décrire la qualité d'un produit ou d'un processus, elle doit être validée avec les critères décrits précédemment. Si une métrique ne réussit pas tous les tests de validité, elle devrait seulement être utilisée selon les critères prescrits par ces tests

(par exemple, si la métrique réussit seulement le test de validité du suivi (« tracking»), elle doit être utilisée seulement pour suivre la qualité d'un produit ou processus).

4 .4.4. Exigences additionnelles

Les exigences additionnelles telles le besoin d'une revalidation, le niveau de confiance dans les résultats analysés et la stabilité de l'environnement sont décrites dans les prochaines sous-sections.

La nécessité d'une revalidation

Il est important de revalider une métrique prédictive avant qu'elle ne soit utilisée pour un autre environnement ou une autre application. Comme pour le processus de génie logiciel qui évolue, la validité des métriques évolue. Les valeurs cumulatives de validation des métriques peuvent être trompeuses puisqu'une métrique qui a été valide pour plusieurs utilisations peut devenir invalide.

Les énoncés d'avertissement suivants devraient être notés :

- une métrique validée n'est pas nécessairement valide dans d'autres environnements ou applications futures ;
- une métrique qui s'est avérée non valide peut être valide dans d'autres environnements ou applications futures.

Confiance dans les résultats analysés

La validation des métriques est un processus continuel. La confiance dans les métriques augmente avec le temps puisque les métriques sont validées sur une variété de projets et que la base de données des métriques et la grosseur de l'échantillon augmentent. Si une métrique est valide, la confiance augmente avec son utilisation (c'est-à-dire, le coefficient de corrélation sera significatif pour des valeurs décroissantes du niveau de signification). La confiance la

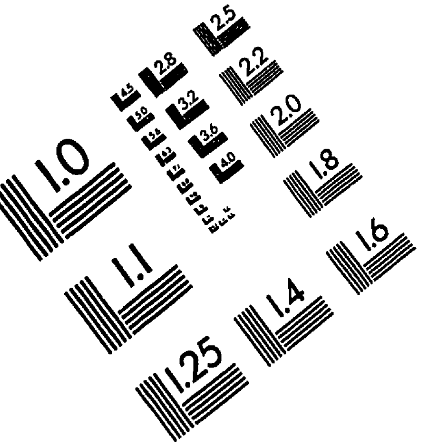
plus élevée survient lorsque les métriques ont été expérimentalement validées en se basant sur les données cueillies dans les projets antérieurs. Même dans ce cas, l'analyse de validation doit continuer.

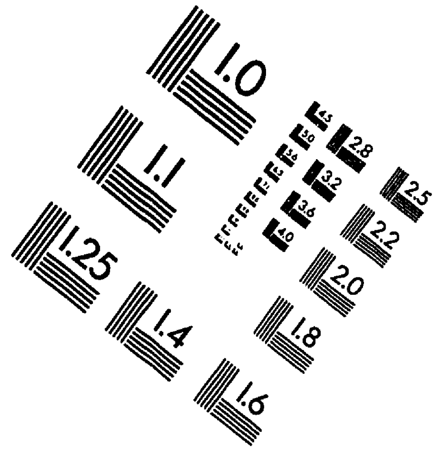
Stabilité de l'environnement

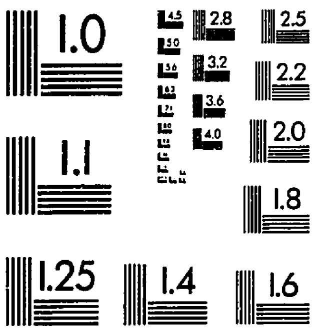
La validation des métriques devrait être entreprise dans un environnement de développement stable (c'est-à-dire, où le langage de design, le langage d'implantation ou les outils de développement ne changent pas durant la vie du projet à partir duquel la validation est faite). De plus, il devrait y avoir au moins un projet dans lequel des données de métriques ont été cueillies et validées avant l'application des métriques. Ce projet devrait être similaire à celui où les métriques sont appliquées selon les compétences en logiciel, la grosseur de l'application et l'environnement logiciel.

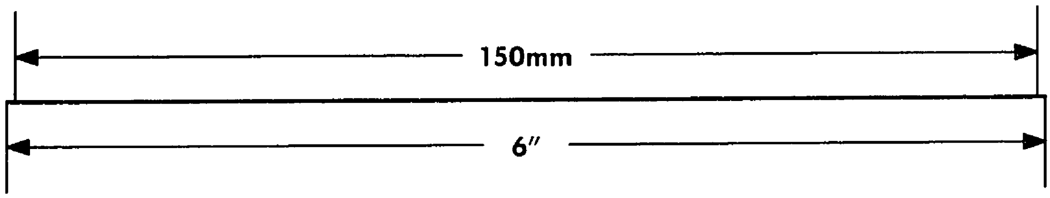
La validation et l'utilisation des métriques devraient être faites durant les mêmes phases du cycle de vie sur différents projets.

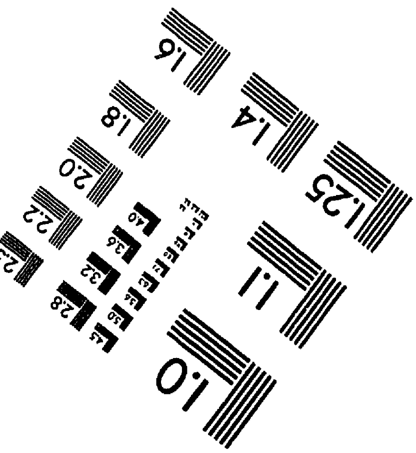
IMAGE EVALUATION TEST TARGET (QA-3)

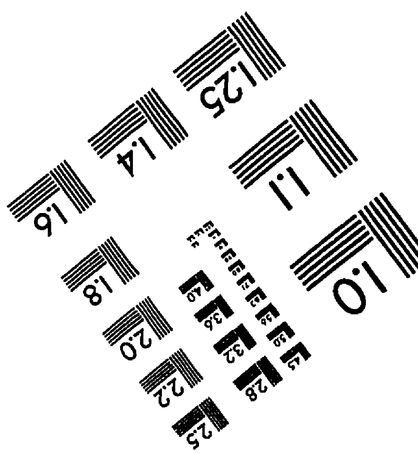












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989