

RÉPUBLIQUE DU CAMEROUN
PAIX-TRAVAIL-PATRIE

UNIVERSITÉ DE DSCHANG

ÉCOLE DOCTORALE



REPUBLIC OF CAMEROON
PEACE-WORK-FATHERLAND

UNIVERSITY OF DSCHANG

POSTGRADUATE SCHOOL

DSCHANG SCHOOL OF SCIENCES AND TECHNOLOGY

Unité de Recherche d'Informatique Fondamentale,
Ingénierie et Appliquée (URIFIA)

SUJET :

APPROCHE D'INTELLIGENCE ARTIFICIELLE POUR
L'ÉVALUATION DE LA QUALITÉ DU LOGICIEL

Mémoire soutenu publiquement en vue de l'obtention du diplôme de
Master en Informatique

OPTION : **INFORMATIQUE FONDAMENTALE**
SPÉCIALITÉ : **RÉSEAUX ET SERVICES DISTRIBUÉS**

PAR :

NKEUGA NGUELIEKAM STEPHANE

MATRICULE : CM-UDS-18SCI3207

Licence en Informatique

SOUS LA DIRECTION DE :

SOH MATHURIN

(Chargés de cours, Université de Dschang)

Année académique : 2019/2020

DÉDICACE

À mon/mon... :

NOM Prenom, petit mot doux;-).

REMERCIEMENTS

TABLE DES MATIÈRES

DÉDICACE	i
REMERCIEMENTS	ii
TABLE DES FIGURES	vi
LISTE DES TABLEAUX	vii
RÉSUMÉ	viii
ABSTRACT	ix
INTRODUCTION GÉNÉRALE	1
Contexte et motivations	1
Problématique	2
Objectifs général et spécifiques	2
Méthodologie	3
Plan du document	3
CHAPITRE 1 ÉTAT DE L'ART.	5
1.1 Introduction	5
1.1.1 Paradigme de la qualité	5
1.1.2 Qualité du logiciel vue sous plusieurs aspects	7
1.1.3 Modèles d'évaluation de la qualité	9
1.2 Evaluation de la qualité Logicielle	17
1.2.1 Évaluation avec les métriques	17
1.2.2 Les métriques de code conventionnelles	18
1.3 Définition et Historique du Machine Learning	27
1.3.1 Définition	27
1.3.2 Historique du Machine Learning	28
1.3.3 Performance et sur-apprentissage	31
1.3.4 Les différents types de machine Learning	32
1.4 Les différents type d'algorithme de Machine Learning	33
1.4.1 Le classifieur naïf de Bayes	33
1.4.2 Les k plus proches voisins	34
1.4.3 Les arbres de décision	34
1.4.4 L'Algorithme AdaBoost	37
1.4.5 Méthode fondée sur l'arbre, J48	38
1.4.6 Random Forest	38
1.5 Prédiction de la qualité du logiciel , travaux menés et méthodes employées	40
1.5.1 Quelques travaux importants dans le domaine de la prédiction . .	41

1.5.2	QUANTIFICATION DES PARAMÉTRES DE QUALITÉ	41
1.5.3	PRÉVISION DE LA QUALITÉ DES LOGICIELS BASÉE SUR L'APPRENTIS- SAGE MACHINE	43
CHAPITRE 2	PRÉ-TRAITEMENT DES DONNÉES ET CONSTRUCTION DU JEU DE DONNÉES	45
2.1	Introduction	45
2.2	Caractéristiques à évaluer	45
2.3	Extraction du programme	53
2.3.1	Attributs et modules des ensembles de données	54
CHAPITRE 3	IMPLÉMENTATION :PRÉSENTATION DES DIFFÉRENTS OUTILS ET IN- TERPRÉTATION DES RÉSULTATS	56
3.1	Introduction	56
3.2	Présentation du modèle de qualité	56
3.3	Présentation de l'extracteur des métriques CK(Chidamber-Kemerer metrics)	57
3.3.1	Fonctionnement CK(Chidamber-Kemerer metrics)	57
3.3.2	Utilisation de CK(Chidamber-Kemerer metrics)	58
3.4	WEKA(Waikato Environment for Knowledge Analysis)	59
3.4.1	Présentation de WEKA	60
CONCLUSION GÉNÉRALE		64
BIBLIOGRAPHIE		66

TABLE DES FIGURES

1.1	Triangle de McCall	10
1.2	Model Mccall	11
1.3	Modèle de Boehm	12
1.4	Le Modèle de qualité de la norme ISO 9126	13
1.5	Modèle SQuaRE	14
1.6	Modèle générique de la Drôme [Dro95]. Le diagramme montre toutes les relations potentielles. Les flèches pleines sont importantes pour le modèle.	15
1.7	Modèle GQM	17
1.8	le processus typique du Machine Learning	28
1.9	Le sur-apprentissage : le graphe montre l'évolution de l'erreur commise sur l'ensemble de test par rapport à celle commise sur l'ensemble d'apprentissage, les deux erreurs diminuent mais dès que l'on rentre dans une phase de sur-apprentissage, l'erreur d'apprentissage continue de diminuer alors que celle du test augmente.	31
1.10	La ligne verte représente un modèle surentraîné et la ligne noire représente un modèle régularisé. Ce dernier aura une erreur de test moins importante.	31
1.11	La validation croisée.	32
1.12	Le classifieur naïf de Bayes est basé sur le théorème de Bayes avec une indépendance (dite naïve) des variables prédictives.	33
1.13	Pour $k = 3$ la classe majoritaire du point central est la classe B, mais si on change la valeur du voisinage $k = 6$ la classe majoritaire devient la classe A	34
1.14	L'ensemble d'apprentissage contient 12 observations décrites par 10 variables prédictives et une variable cible. [1]	35
1.15	Les exemples positifs sont représentés par des cases claires alors que les exemples négatifs sont représentés par des cases sombres. (a) montre que la division par l'attribut Type n'aide pas à avoir une distinction entre les positifs et les négatifs exemples. (b) montre qu'avec la division par l'attribut Patrons on obtient une bonne séparation entre les deux classes. Après division par Patrons, Hungry est un bon second choix. [1]	36
1.16	L'arbre de décision déduit à partir des 12 exemples d'apprentissage.[1]	36
1.17	Algorithme AdaBoost	37
1.18	Algorithme de Random Forest	39
2.1	Mapping entre les attributs de qualité et les métriques du code	53
2.2	Schema de décomposition des attributs de qualité	55
3.1	Clonage du l'outil CK	58
3.2	Générer le point jar	59
3.3	Calcul des métriques	59
3.4	Aperçu des fichiers	60
3.5	Structure du logiciel Weka	61

3.6	Interface graphique de WEKA	61
-----	---------------------------------------	----

LISTE DES TABLEAUX

1.1	Liste des propriétés porteuses de qualité. Chacune de ces propriétés peut être appliquée à une ou plusieurs formes structurelles (les expressions, les variables, les boucles), et affecte un ou plusieurs attributs de qualité	16
1.2	Un exemple de graphe de flot de contrôle avec les trois différents chemins linéairement indépendants.	20
1.3	Les moyennes simples de la métrique SLOC pour quatre méthodes dans deux classes différentes	25
1.4	Exemple de poids appliqués sur SLOC	26
1.5	Les moyennes de deux versions différentes d'un même projet	26
1.6	Historique	30
2.1	Attributs de qualité du modèle ISO 9126	46
2.2	Le tableau complet des caractéristiques et sous-caractéristiques du modèle de qualité ISO 9126-1	48
2.3	les caractéristiques et attributs de la maintenabilité	48
2.4	Les caractéristiques et attributs de l'utilisabilité	49
2.5	Les caractéristiques de la fiabilité	50
2.6	Tableau des propriétés conception mesurées	51
2.7	Formules de calcul des attributs de qualité de Bansiya	52

RÉSUMÉ

Mots clés :

ABSTRACT

Keywords :

INTRODUCTION GÉNÉRALE

SOMMAIRE

Contexte et motivations	1
Problématique	2
Objectifs général et spécifiques	2
Méthodologie	3
Plan du document	3

Contexte et motivations

Voici bientôt plus d'un demi-siècle que la vie de l'homme se retrouve bouleverser par les systèmes informatiques et plus particulièrement les logiciels et programmes informatiques, qui sont de plus en plus omniprésent dans nos quotidiens peu importe le secteur d'activité ou l'on se situe finance, éducation, la santé, l'armée pour n'en citer que ceux-la. Cette présence très grandissante a fait passer la notion de qualité du logiciel du stade de besoin de confort vers celui du besoin critique. Avec le temps ces logiciels deviennent plus complexes et universels, mais avant que ces logiciels ne soient mises à la disposition du grand public (utilisateurs finaux) ils se doivent de respecter certains critères de qualité d'un logiciel à savoir la fiabilité, la capacité fonctionnelle ou encore la maintenabilité. L'ISO 9126 définit la qualité logicielle comme étant l'ensemble des traits et caractéristiques d'un produit logiciel portant sur son aptitude à satisfaire des besoins exprimés et implicites (source : norme ISO/IEC 9126 :2001). En effet, toute défaillance d'un système informatique est potentiellement génératrice de désagréments qui peuvent, dans certains cas limites, provoquer de graves atteintes à l'intégrité économique des organisations ou pire à l'intégrité physique des personnes. Pour exemple, à la date du dimanche 10 mars 2019 le crash d'un Boeing 737 Max appartenant la compagnie aérienne Ethiopian Airlines tuant au total 157 personnes. Le crash est causé par un nouveau système logiciel appelé "Maneuvering Characteristics Augmentation System" (MCAS), que la société Boeing a développé pour résoudre les problèmes de stabilité dans certaines conditions de vol induites par les nouveaux moteurs plus puissants de l'avion. Selon le rapport de la commission des transports et des infrastructures de la Chambre des représentants du parlement américain, la société Boeing savait tout le temps que les pilotes n'avaient que 10 secondes pour identifier le problème et le traiter avant d'être dépassés par les actions malveillantes du MCAS. Et lorsque le MCAS s'est déclenché mortellement, dans le cas de ce crash, il réagissait à de fausses données fournies par un capteur situé sur le nez de l'avion, qui suggéraient que l'avion était en train de décrocher, alors que ce n'était pas le cas. Le crash de la navette Ariane 5 en 1996 qui s'est brisée 40 secondes après le décollage à cause d'un bug informatique dans le système de pilotage automatique. Un bogue qui a provoqué une perte de 370 millions de dollars [Mordal, 2012]. Ou encore le site de vente en ligne "Ebay" indisponible pendant 22

heures en 1999, qui entraîna une perte pour le groupe estimé entre 3 et 5 millions de dollars [Karine Mordal,2012].

Problématique

Le problème que nous étudions dans ce mémoire est celui qui consiste à une évaluation de la qualité d'un logiciel avec les techniques d'intelligence artificiel. A partir du contexte nous avons formulé cette problématique en plusieurs questions qui suscitent un très grand intérêt que ce soit au niveau des entreprises productrices de logiciel ou même des utilisateurs finaux à qui sont destinées ces logiciels comme suit : qu'est-ce que la qualité ? qu'est-ce que la qualité logicielle ? quelles sont les caractéristiques ou attributs d'un logiciel de qualité ? comment peut-on évaluer cette qualité logicielle avec l'apprentissage automatique ? Ces différentes questions feront l'objet d'une étude poussée tout au long de ce travail.

Objectifs général et spécifiques

L'objectif général de ce mémoire est principalement focalisé sur l'évaluation de la qualité du logiciel en tant que produit. Pour pouvoir atteindre cet objectif nous allons utiliser les métriques de code statiques et les techniques d'apprentissage automatique (supervisées et non supervisées).

De ce fait évaluer un logiciel permet à la fois d'obtenir une image précise de la qualité de ce dernier mais peut également fournir une indication quant à son comportement dans le temps : quels sont les risques de bogues, les éventuelles failles sécuritaires, les difficultés de maintenance, les freins à l'évolution, la viabilité à long terme, etc. De plus, évaluer la qualité d'un logiciel peut également servir d'outil pour déterminer si le logiciel correspond aux attentes du client.

Un des objectifs spécifiques de la mesure de la qualité logicielle consiste également à sensibiliser les équipes de développement sur leurs méthodes de programmation. En effet, mesurer la qualité a de surcroît pour objectif de formaliser de bonnes pratiques de travail et des indicateurs permettant d'augmenter la qualité des développements. De même, vouloir mettre en place une mesure de la qualité oblige à formaliser les processus de conception au sein d'une entreprise productrice de logiciel, en commençant par l'expression la plus précise possible des besoins, jusqu'aux processus de tests mis en place pour vérifier l'adéquation entre les fonctionnalités du logiciel et ses objectifs. Un autre objectif et pas des moindres est de permettre aux différentes équipes intervenant dans le processus de conception et d'évolution du logiciel de détecter assez rapidement les Bugs ou défauts des logiciels, de les corriger avant la mise en production du logiciel proprement dit permettant ainsi aux entreprises productrices de logiciels de gagner en terme de coût et surtout en temps. Et enfin évaluer la qualité d'un logiciel toujours avant sa mise en production est un moyen plus ou moins fiable de garantir que le logiciel ne pourra porter atteinte ni à l'intégrité financière des organisations ni à celui de personnes utilisant le logiciel.

Méthodologie

Kitchenham cite une liste plus détaillée des aspects de la qualité dans ces [KP]. Cette liste est composée par Garvin [Gra84], qui a résumé ces aspects en points de vue. Il a défini le point de vue de l'utilisateur, qui considère la qualité comme étant adaptée à l'objectif de l'utilisateur ; le point de vue du fabricant, qui considère la qualité comme étant conforme aux spécifications ; le point de vue du produit, qui considère la qualité comme étant liée aux caractéristiques inhérentes du produit ; et le point de vue fondé sur la valeur, qui considère la qualité comme dépendant de ce qu'un client est prêt à payer pour l'obtenir. Notre travail porte sur l'évaluation de la qualité des logiciels à l'aide de mesures du code source et a donc une vision de la qualité du produit. L'informatique a mis au point plusieurs modèles pour décrire la qualité des produits logiciels. L'ISO a essayé de consolider les différentes vues sur la qualité dans un modèle de qualité général [iso06]. Des modèles comme la norme ISO donnent une bonne idée de ce que signifie la qualité des logiciels dans l'industrie. Malheureusement, ces directives fonctionnent à un niveau d'abstraction élevé et sont difficiles à évaluer pour les différents artefacts logiciels. Un autre problème est l'objectivité de l'estimation de la qualité. Comment peut-on évaluer objectivement un concept de qualité abstrait tel que la maintenabilité ou la fiabilité. Les mesures du code source promettent un moyen objectif et automatisé de recueillir des informations sur le produit logiciel, le plus important du développement de logiciels étant : le code source.

Nous analysons plusieurs modèles de qualité existants afin d'évaluer l'utilité des mesures du code source pour exprimer la qualité sur leurs niveaux d'abstraction spécifiques en précisant quelques avantages et inconvénients associés à ces modèles. Nous relions également les métriques du code source aux attributs de qualité de haut niveaux, mais néanmoins dans un contexte de développement spécifique, les métriques de code source peuvent fournir des informations utiles.

En outre, nous expliquons comment l'utilisation de mesures du code source peut aider à évaluer la qualité des logiciels. Chidamber et Kemerer ont mis au point des mesures de code source pour mesurer ce qui se passe dans le code [CK94, BBM96]. Nous évaluons la qualité du code à un niveau d'abstraction plus élevé en nous basant sur des combinaisons de mesures du code source. Notre outil d'évaluation de la qualité résume les mesures pour exprimer les attributs de qualité qui fournissent des informations sur la qualité du code source et donc sur l'état interne du produit. Ces mesures serviront d'entrée à notre modèle d'apprentissage automatique

Plan du document

Ce mémoire comporte 4 chapitres

A l'introduction générale nous décrivons le contexte, les motivations, la problématique, les objectifs généraux et spécifiques et notre contribution. Au chapitre 1 nous présentons l'état de l'art en ce qui concerne la qualité du logiciel, son évaluation, en passant par une brève présentation des différentes techniques d'apprentissage automatique, les algorithmes d'apprentissage automatique utilisés dans la prédiction de la qualité logicielle et enfin les travaux connexes

ayant déjà traites de ce sujet.

Au chapitre 2 fait état du pré-traitement et la construction du jeu de données, dans ce chapitre nous commençons tout abord par identifier les différentes caractéristiques de la qualité des sous-caractéristiques et métriques associées à ces dernières , puis faisons le choix des caractéristiques à évaluer suivant le modèle qualité ISO-9126 , puis vient l'étape d'extraction des données du logiciel à évaluer : cette extraction des données ce fait essentiellement à partir des métriques de Chidamber et de Kemerer qui sont des métriques de code procéduraux et orientés objets, enfin vient la formation du jeu de données pour la prédiction de la qualité à partir d'une combinaison des mesures des métriques retenues pour la prédiction.

Au chapitre 3 traite de l'implémentation, nous présentons le modèle de qualité mise en ?uvre pour l'évaluation de la qualité, le choix des environnements, leurs fonctionnement et leur utilisation.

Au chapitre 4 nous avons les résultats expériences et analyses, ici nous faisons une description es jeux des données, ensuite appliquons les différents algorithmes respectivement avec une validation croisée, puis avec un échantillonnage de (80,20) c'est-à-dire 80% du jeu de données pour la l'apprentissage et 20 % pour la validation et enfin sur l'ensemble du jeu de données. Dans la conclusion générale nous présentons les limites et les perspectives de notre travail.

Chapitre 1 : État de l'art. Dans ce chapitre, nous présenterons tout d'abord les gourous de la qualité, par la suite nous définirons la qualité du logiciel sous plusieurs aspects. Dans ce chapitre nous abordons également les modèles de qualité du logiciel qui sont les plus évoques dans la littérature. Enfin nous passons en revue les algorithmes d'apprentissage automatique pour l'évaluation de la qualité du logiciel et également les travaux qui se sont penchés sur l'évaluation de la qualité avec une approche d'intelligence artificielle.

Chapitre 2 : Pré-traitement des données et construction du jeu de données Dans ce chapitre, nous nous concentrons sur la formation du data-set qui servira d'entrée au système d'apprentissage automatique. Pour l'obtention de ce data-set nous servons des métriques de code extraites dans le code source du logiciel a partir d'une analyse statique du code.

Chapitre 3 : Implémentation :Présentation des différents outils Et interprétation des résultats Dans ce chapitre, nous présentons les différents outils que nous utilisons, les environnements leurs fonctionnement et utilisation et enfin les résultats

Conclusion générale. Nous y dressons un bilan de notre travail en faisant mention des difficultés rencontrées, et répertorions quelques perspectives d'amélioration de ce dernier.

ÉTAT DE L'ART.

SOMMAIRE

1.1	Introduction	5
1.2	Evaluation de la qualité Logicielle	17
1.3	Définition et Historique du Machine Learning	27
1.4	Les différents type d'algorithme de Machine Learning	33
1.5	Prédiction de la qualité du logiciel , travaux menés et méthodes employées	40

1.1 Introduction

1.1.1 Paradigme de la qualité

Tout d'abord nous essayer de donner une définition appropriée au terme "qualité" notamment la qualité logicielle

1.1.1.1 Les Gourous de la qualité

Le terme qualité est un terme éminemment ambigu qui comporte plusieurs significations et implique des sens différents selon le contexte. Ainsi d'après certains acteurs dans la littérature appelé "gourous" de la qualité on distingue la qualité selon :

Deming

Deming accorde une place importante et une grande responsabilité à la gestion, aussi bien au niveau de l'individu que de l'entreprise. Pour lui, le contrôle est responsable de 94% des problèmes de la qualité. Il dresse en quatorze points une méthodologie pour la gestion de la qualité. Elle est applicable par des petits ou des grands organismes des secteurs publics et privés. Sa méthodologie concerne aussi bien les problèmes organisationnels, de production, de gestion ou de maintenance. Pour Deming, la qualité, c'est la satisfaction du besoin des clients ou consommateurs [Deming, 1986].

Crosby

L'approche de Crosby est le zéro défaut (zero defect) [2]. Elle ne signifie pas que les erreurs ou les défauts ne sont pas acceptables. Les organisations ne doivent pas réfléchir en termes d'erreurs à venir ni planifier d'éventuels ajustements. Son approche se traduit en quatre

points :

La qualité est la conformité à la spécification des besoins [2]. Ces besoins doivent être clairement définis, l'objectif étant la satisfaction des besoins du client.

Le système de qualité est la prévention. La prévention avant l'évènement est plus efficace et rentable que sa détection après.

La performance standard est d'atteindre le niveau de « zéro défaut ». L'objectif ne doit être rien d'autre qu'une qualité parfaite et les coûts de la prévention ne doivent pas croître de façon exponentielle à l'approche de l'état de « non-défaut » (zéro défaut) du produit. La qualité ne doit pas avoir un coût excessif.

La mesure de la qualité est le prix de la non-conformité aux spécifications.

Le coût des erreurs est le premier facteur de motivation et si les efforts vont dans le sens de leur prévention, il en ressortirait une meilleure production, moins de réajustements de travail et une meilleure satisfaction des besoins du client. Il propose également une méthodologie en quatorze étapes pour la mise en l'œuvre d'un processus de gestion de la qualité dans une entreprise ou organisation.

Taguchi

Pour Taguchi, la qualité et la fiabilité doivent être effectives dès le niveau conceptuel, le but étant de construire ou fabriquer des produits qui sont (seront) insensibles aux variations pouvant apparaître dans n'importe quelle étape de leur cycle de vie. Il s'oppose ainsi à l'inspection ou à la recherche des défauts après fabrication. Taguchi dans ses travaux [Taguchi, 1986, Taguchi and Wu, 1980] développe une fonction de perte pour la qualité. Cette fonction est quadratique et en association avec la déviation (variation) sur la meilleure qualité attendue sur une caractéristique (propriété) donnée. Ainsi, cette variation de la fonction de perte est inversement proportionnelle à l'augmentation de la qualité [Taguchi, 1986].

Suite aux approches de ces « gourous » de la qualité, celle-ci reste quelque chose de floue. Cependant, il ressort qu'elle est l'ensemble des critères fixés par une organisation pour la satisfaction du client par rapport à des produits ou services qu'elle propose. Nous présenterons ici un résumé des différents gourous de la qualité :

1. La qualité est la conformité à des spécifications ou exigences [Crosby, 1979].
2. La qualité est la satisfaction du besoin des clients ou consommateurs [Deming, 1986].

Mais pour l'ingénierie du logiciel, qu'est-ce que la qualité logicielle ? La notion de qualité est souvent subjective : elle varie en fonction des exigences, du métier, de la relation que l'on a avec le logiciel, du type même de logiciel. La qualité est-elle définie de la même manière selon que l'on soit le client (celui pour qui le logiciel a été conçu), l'utilisateur (celui qui se sert du logiciel au quotidien), le développeur (celui qui écrit les lignes de code), le testeur (celui qui valide le logiciel), ou encore le manager. Prenons tout d'abord les différentes définitions de la qualité logicielle :

1. la norme ISO 9001 définit la qualité comme le degré auquel un ensemble de caractéristiques remplit les exigences (du client) [ISO/IEC, 2008] ;
2. le livre swebok donne la définition suivante : un ensemble de règles et de principes à suivre au cours du développement d'une application afin de concevoir un logiciel répondant aux attentes (du client), le tout sans défaut d'exécution [Abran et al., 2004] ;

3. la norme ISO 9126 définit la qualité logicielle comme étant la capacité à satisfaire les besoins exprimés et implicites [ISO/IEC, 2001] ;

Ces différentes définitions mettent en avant les notions suivantes :

- exprimer des besoins implicites et explicites ;
- remplir des exigences, répondre aux attentes, être capable de satisfaire des besoins ;
- formaliser des ensembles de règles et principes à suivre ;
- garantir qu'il n'y a pas de défaut d'exécution.

A partir de ces différentes considérations, les définitions et les principes suivants peuvent être posés : La qualité d'un logiciel est le degré auquel un logiciel remplit les besoins et exigences des clients sans défaut d'exécution. La qualité logicielle s'exprime sous forme de règles et de principes à suivre pour développer un logiciel capable de répondre aux attentes du client, c'est-à-dire développer un logiciel de qualité. Les besoins d'un client doivent être exprimés avec soin, de manière précise et détaillée et doivent faire l'objet d'une étude minutieuse. La qualité d'un logiciel repose donc sur deux notions essentielles : les attentes du client et l'absence de défaut.

1.1.2 Qualité du logiciel vue sous plusieurs aspects

La qualité est en général une notion ambiguë : cette ambiguïté réside dans la multitude de ses définitions employées dans chaque domaine. Selon Kan [1995], cela peut être attribué à plusieurs facteurs car, la qualité est multidimensionnel. Elle dépend, en fait, de la nature de l'entité étudiée, de son environnement et de ses attributs. De plus, le terme «qualité» fait partie de notre langage quotidien et sa popularité engendre plusieurs connotations (compréhensions) du terme qui peuvent être différentes de son utilisation professionnelle et scientifique. Différentes dimensions de la qualité sont résumées par Garrin [1984] en cinq vues à partir desquelles il décrit la qualité :

- la vue transcendantale : la qualité est quelque chose qu'on peut reconnaître, mais qu'on ne peut pas définir ;
- la vue d'utilisateur : la qualité est la force apparente du produit pour réaliser des fonctions ;
- la vue de fabrication : la qualité est la conformité aux spécifications
- la vue du produit : la qualité est attachée aux caractéristiques intrinsèques du produit. c'est la vue la plus évoquée par les experts de la qualité du logiciel
- la vue basée sur la valeur : la qualité dépend des coûts du produit

Dans le domaine des logiciels, cette confusion est limitée. Cependant, la qualité d'un logiciel est souvent définie dans un sens étroit comme l'absence de « bogues » dans le produit logiciel. Cette définition rejoint la notion de conformité, car un logiciel ayant des défauts est non conforme aux spécifications, mais elle ne mesure pas la satisfaction du client vis-à-vis de certains aspects, comme la facilité d'utilisation et de maintenance, la clarté de la documentation, etc. Une définition claire et juste de la qualité demeure encore un souci pour la communauté du génie logiciel. Deux voies de recherche sur la qualité sont généralement empruntées. Dans la première on s'occupe de la qualité du processus de développement du logiciel et on croit que la qualité de ce dernier détermine la qualité du produit logiciel. Dans la deuxième on juge qu'un bon processus ne garantit pas un bon produit et que la qualité de ce dernier est plutôt

déterminée par un ensemble de propriétés intrinsèques contenues dans le produit.

Selon une troisième perspective, Kitchenham et Pfleeger [1996], en analysant la question « qu'est ce qui distingue un bon logiciel d'un mauvais », soulignent que le contexte aide à déterminer la réponse. En effet, les fautes tolérées dans un logiciel de traitement de textes ne sont pas acceptées dans un système de contrôle de sécurité d'un avion. Par conséquent. Ils suggèrent que la qualité doit être considérée au moins de trois façons : la qualité du produit, la qualité du processus qui engendre le produit et la qualité du produit dans le contexte d'environnement où il sera utilisé.

Qualité du produit

Si on demande à différentes personnes d'indiquer les caractéristiques d'un produit logiciel qu'elles jugent importantes pour sa qualité, il est très probable que leurs réponses soient différentes. Cela est dû au fait que l'importance des caractéristiques dépend du profil de la personne qui analyse le produit. En effet, l'utilisateur croit que la qualité réside dans la capacité du logiciel de faire ce qu'il veut et d'une manière simple (facilité d'utilisation). Il privilégie ainsi, les caractéristiques externes du produit. Cependant, un produit logiciel est jugé aussi par d'autres acteurs, comme celui qui le conçoit, celui qui l'implémente et celui qui fait sa maintenance après le codage et pendant la mise en opération. Ces acteurs voient la qualité à travers des caractéristiques internes du produit logiciel, avant même son achèvement [Pfleeger. 1998].

Des modèles de qualité sont construits pour relier certaines caractéristiques internes avec des caractéristiques externes qui peuvent représenter la vue de l'utilisateur ou celle du développeur du logiciel. Par exemple. le modèle McCall. construit par McCall et ses collègues en 1977. montre comment les facteurs externes de la qualité sont reliés aux critères de la qualité de produit [McCall et Walters. 1977]

Le modèle ISO 9126 est un standard de la qualité du produit grandement utilisé depuis sa création au début des années 80. Il est inspiré du modèle de McCall. En plus de l'organisation hiérarchique des différentes caractéristiques de la qualité du produit, le modèle ISO 9126 offre un ensemble de définitions et de termes usuels pour l'évaluation de la qualité du produit logiciel [ISO. 2003]. Plus de détails sur le standard ISO 9126 sont donnés dans la suite de ce mémoire

Qualité du processus

L'intérêt pour la qualité du processus découle du fait qu'un produit logiciel peut rencontrer des problèmes sérieux, parce que certaines activités lors de son développement, sont négligées ou mal exécutées. Plusieurs chercheurs croient que la qualité du processus de développement et celui de maintenance est aussi importante que la qualité du produit. L'avantage d'améliorer la qualité d'un processus est de pouvoir améliorer en même temps la qualité des produits logiciels. De la même manière que pour la qualité du produit, des modèles d'amélioration du processus logiciel et d'évaluation de ses capacités sont proposés, comme CMM (Capability Maturity Model) et SPICE (Software Process Improvement and Capability Determination) [Pressman, 1996].

Qualité dans un contexte d'environnement commercial

Ce niveau de la qualité du logiciel est proposé par Kitchenham et Pfleeger [1996]. Il essaie de montrer un autre côté du logiciel rarement analysé, à savoir, la qualité des services que le logiciel fournit dans le milieu économique où il est exploité. Il s'agit de la valeur économique d'un logiciel [Simrnons, 1996]. Cette proposition vient de l'idée qui suppose que l'amélioration technique (la qualité technique du processus ou du produit) est traduite en valeurs économiques. plus de détails sont disponibles dans [Pllee2er. 1998]

1.1.3 Modèles d'évaluation de la qualité

Evaluer la qualité consiste à définir des règles, interpréter des indicateurs de qualité en fonction de ces règles, leur donner du sens et les utiliser pour qualifier des concepts qualitatifs. Pour y parvenir, les entreprises utilisent des modèles de qualité. Ces modèles définissent des concepts, des domaines et des règles qualitatifs qu'ils évaluent via les mesures et métriques effectuées sur le système. La pertinence d'un modèle de qualité est fondée sur sa capacité à modéliser, qualifier, évaluer un certain nombre de règles complexes à partir de mesures (comme par exemple évaluer la structure générale du code source à partir de métriques de code). Nous présentons les principaux modèles de qualité existants actuellement. Ce sont des modèles hiérarchiques qui recensent les principes de qualité en partant des exigences globales et des principes les plus généraux pour descendre vers les métriques qui permettent de les mesurer. Ceci implique que la mesure de la qualité ne peut débiter qu'une fois le modèle totalement spécifié et que les premiers résultats ne peuvent être obtenus qu'une fois la collecte des données suffisante.

1.1.3.1 Le modèle de McCall : Facteurs Critères Métriques-FCM : Qualité logiciel orienté produit

Le modèle de qualité de McCall introduit en 1977 [MRW77] est l'un des premiers modèles de ce type. Il est axé en priorité sur les développeurs et le processus de développement. En choisissant des facteurs de qualité des logiciels, qui reflètent le point de vue de l'utilisateur et du développeur, McCall et al... tente de combler le fossé entre ces deux parties prenantes. Le modèle de McCall est un modèle hiérarchique typique, basé sur les catégories. Au niveau supérieur, nous avons trois grandes perspectives. La perspective de la révision du produit, dans un premier temps, définit la capacité du produit logiciel à subir des changements. Ensuite, la perspective de transition du produit représente l'adaptabilité du logiciel à de nouveaux environnements et, enfin, les opérations du produit représentent les caractéristiques des opérations du logiciel. Chacune de ces trois catégories comprend plusieurs facteurs de qualité :

- **Révision du produit :**
- **Maintenabilité :** l'effort nécessaire pour localiser et réparer une défaillance du programme dans son environnement de fonctionnement
- **Flexibilité :** la facilité d'apporter les changements requis par les modifications de l'environnement opérationnel
- **Testabilité :** la facilité de tester le programme, afin de s'assurer qu'il est exempt d'erreurs et qu'il répond à ses spécifications
- **Transition des produits :**

- Portabilité : l'effort nécessaire pour transférer un programme d'un environnement à un autre
- Réutilisation : la facilité de réutiliser un logiciel dans un contexte différent
- Interopérabilité : l'effort nécessaire pour coupler le système à un autre système
- **Opérations sur les produits :**
- Correction : la mesure dans laquelle un programme est conforme à sa spécification
- Fiabilité : la capacité des systèmes à ne pas tomber en panne
- Efficacité : subdivisée en efficacité d'exécution et en efficacité de stockage, et signifiant généralement l'utilisation des ressources, par exemple le temps de processeur, le stockage
- Intégrité : la protection du programme contre les accès non autorisés
- Facilité d'utilisation : la facilité du logiciel

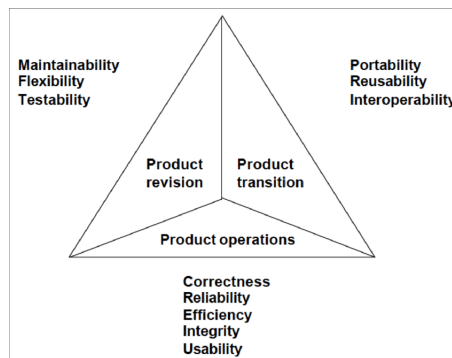


FIGURE 1.1 – Triangle de McCall

Dans le modèle de McCall, 23 critères de qualité sont également définis. Ces critères sont les attributs d'un ou de plusieurs facteurs de qualité. Des mesures sont utilisées afin de quantifier certains aspects des critères. Les mesures de qualité sont obtenues en répondant à un certain nombre de questions "oui" ou "non". En fonction de la réponse donnée, la qualité est évaluée. Le modèle de McCall a été critiqué parce que le jugement de la qualité est mesuré subjectivement, sur la base du jugement de la personne qui répond aux questions.

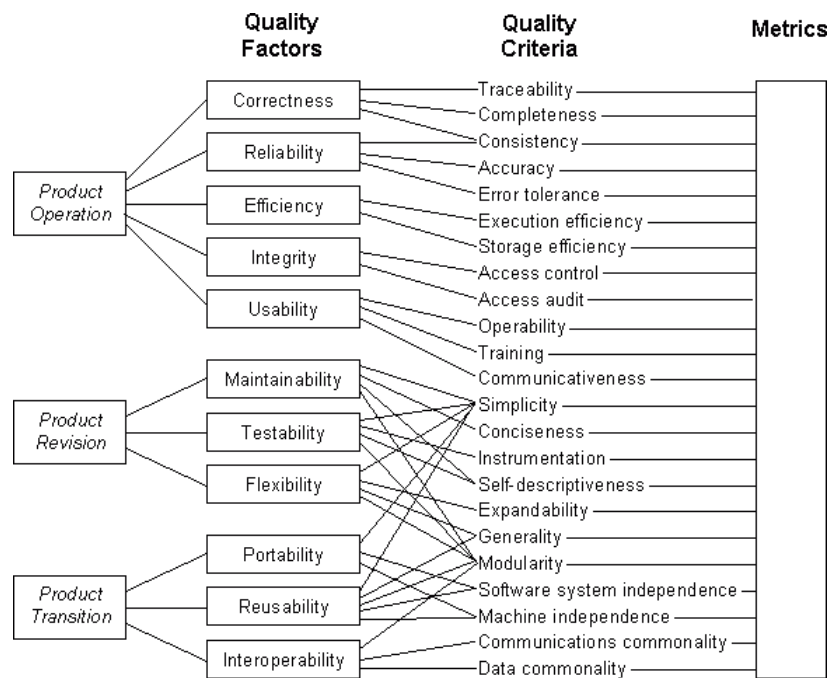


FIGURE 1.2 – Model McCall

1.1.3.2 Boehm, qualité des produits logiciels

Un autre modèle de qualité a été introduit par Boehm [BBK+78] en 1978. Il s'agit également d'un important pré-décesseur des modèles de qualité actuels. Boehm prend en compte les lacunes contemporaines des modèles qui évaluent automatiquement et quantitativement la qualité des logiciels. Fondamentalement, son modèle tente de définir qualitativement la qualité des logiciels par un ensemble donné d'attributs et de mesures. Il existe certains parallèles reconnaissables entre le modèle de McCall et le modèle de Boehm. Par exemple, tous deux proposent un modèle hiérarchique structuré avec des caractéristiques de haut niveau, de niveau intermédiaire et de bas niveau. Toutes ces caractéristiques influencent les niveaux de qualité supérieurs. Les caractéristiques de haut niveau de la hiérarchie de la qualité de Boehm comportent trois caractéristiques de haut niveau qui répondent aux trois principales questions que peut se poser un acheteur de logiciels.

- Utilité en l'état : dans quelle mesure (facilement, sûrement, efficacement) puis-je l'utiliser en l'état ?
- Maintenabilité : Est-il facile de comprendre, de modifier et de tester à nouveau ?
- Portabilité : Puis-je toujours l'utiliser si je change d'environnement ?

Au niveau intermédiaire, il existe 7 facteurs de qualité qui représentent les qualités attendues d'un système logiciel :

- Portabilité : Le code peut être utilisé facilement et correctement dans d'autres environnements.
- Fiabilité : Le code remplit ses fonctions de manière satisfaisante.
- L'efficacité : Le code exécute son intention sans gaspillage de ressources.
- Utilisabilité : Le code est fiable, efficace et conçu dans le respect de l'homme.

1.1. INTRODUCTION

- Testabilité : Le code facilite la mise en place de critères de vérification et permet d'évaluer ses performances.
- La compréhensibilité : Le code est facile à lire dans le sens où les inspecteurs peuvent rapidement reconnaître son utilité.
- Flexibilité : Le code est facile à modifier, lorsqu'un changement souhaité a été déterminé.

Au niveau inférieur du modèle, il existe des hiérarchies de métriques de caractéristiques primitives. Ces caractéristiques constituent la base de la définition des métriques de qualité. Construire une telle base était l'un des objectifs que Boehm voulait atteindre. Le modèle propose donc au moins une métrique, qui devrait mesurer une caractéristique primitive donnée. Boehm a défini la métrique comme "une mesure de l'étendue ou du degré auquel un produit possède et présente une certaine caractéristique (de qualité)". La figure 1.3 montre les

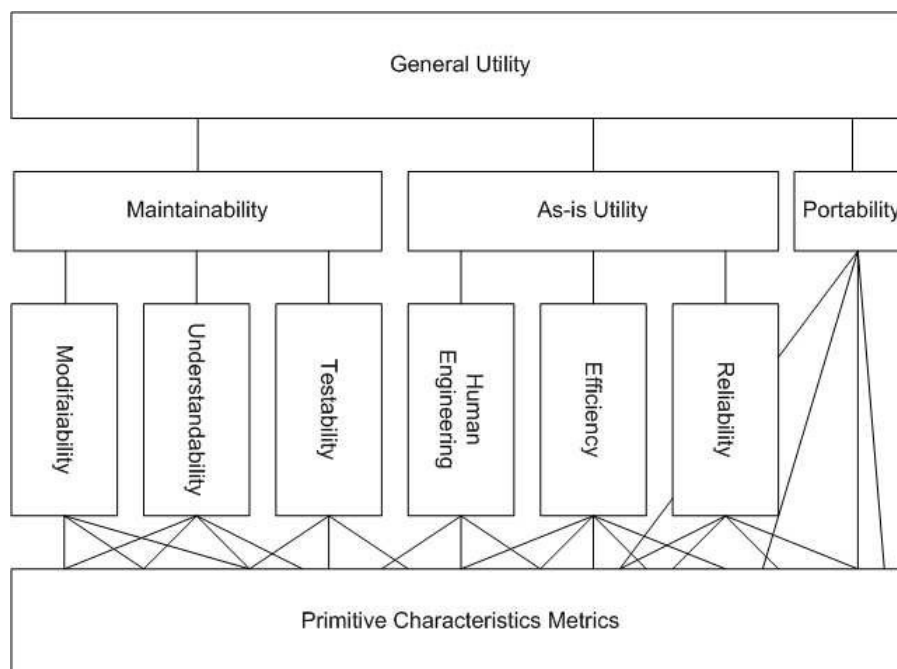


FIGURE 1.3 – Modèle de Boehm

caractéristiques de haut niveau (utilité telle quelle, maintenabilité, portabilité), qui sont nécessaires à l'utilité générale. Au niveau inférieur, on trouve les mesures caractéristiques comme l'indépendance de l'appareil, l'autonomie, la précision, l'exhaustivité, la robustesse/intégrité, la cohérence, la responsabilité, l'efficacité de l'appareil, l'accessibilité, la communication, l'autodescription, la clarté, la lisibilité et la capacité d'augmenter. La plus grande différence entre Boehm et McCall est que le modèle de Boehm repose sur un large éventail de caractéristiques de qualité, avec un accent particulier sur la maintenabilité. McCall, en revanche, se concentre davantage sur la mesure précise de la propriété de haut niveau "As-is Utility".

1.1.3.3 La norme ISO-9126

Le modèle défini par la norme ISO-9126 décrit un ensemble de caractéristiques et de sous-caractéristiques liées à la qualité d'une application, à la fois internes et externes. Ce modèle considère six caractéristiques de base (fiabilité, efficacité, etc.) figure 1.4, subdivisées en

1.1. INTRODUCTION

sous-caractéristiques. Les six caractéristiques du modèles ISO-9126 sont composées d'attributs mesurables ou non.

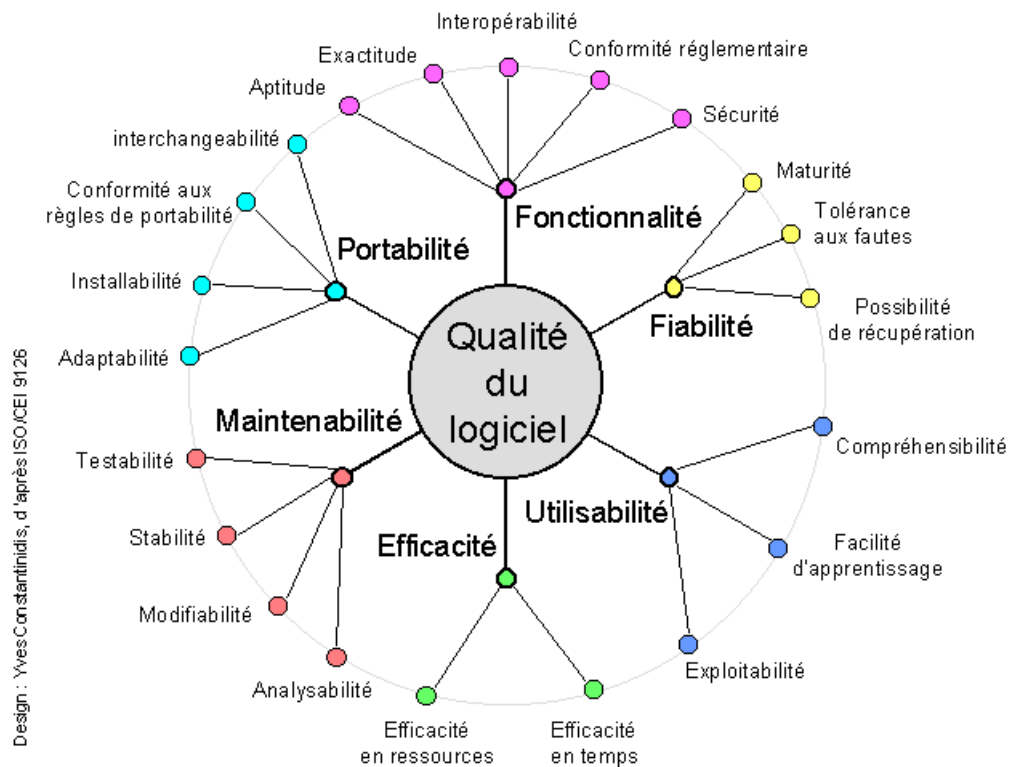


FIGURE 1.4 – Le Modèle de qualité de la norme ISO 9126

Ce modèle tire son origine des modèles de [McCall J. Richards P. Walters G., 1977] et de [Boehm et al., 1978]. C'est un modèle générique qui permet aux usagers de développer leurs propres critères. Ce modèle n'atteint pas le niveau des métriques. Il laisse aux utilisateurs le choix des métriques à implémenter dans leur modèle. ISO est également à l'origine d'autres normes et modèles sur les logiciels basés sur l'aspect gestion des processus de développement et d'évolution de la qualité. Comme nous le verrons plus bas, les trois aspects de la qualité du logiciel sont reliés deux à deux par la complémentarité. Les évolutions futures de ISO 9126 comme SQUARE (Software Product QUALity Requirement and Evaluation [W. Suryn, 2003] tiennent compte et intègrent ces évolutions. SQUARE ajoute également l'évaluation de la qualité obtenue. SQUARE préconise les quatre étapes suivantes :

- fixer les exigences de la qualité, comme par exemple les spécifications à respecter ;
- établir un modèle de qualité, par le choix d'un modèle de qualité adapté aux besoins de l'utilisateur ;
- fixer les métriques de qualité, le choix judicieux des métriques intervenant dans le processus d'évaluation de la qualité ;
- conduire des évaluations.

1.1.3.4 SQuaRE, Software product QUALity Requirement and Evaluation

La norme SQuaRE définie depuis 2005 est la norme qui succède au standard ISO 9126. Elle a été définie à partir de ISO 9126 et de la partie évaluation de la norme ISO 14598 [ISO/IEC, 2005]. Elle a pour objectif d'intégrer ces deux normes pour n'en former qu'une. Elle

veut également répondre aux différents besoins de la qualité selon les acteurs (développeurs, testeurs, utilisateurs, clients). De plus, elle unifie les différents documents normatifs autour de la qualité. SQuaRE définit quatre étapes pour permettre une meilleure évaluation de la qualité : définir les exigences, établir un modèle de qualité, définir les métriques de la qualité et évaluer la qualité.

Différences par rapport à ISO 9126

La norme SQuaRE reprend le modèle défini dans ISO 9126 et y apporte les changements suivants :

- La portée du modèle de qualité a été étendue pour y inclure la qualité à l'utilisation en tant que modèle à part entière (il reste cependant le facteur facilité d'utilisation dans le modèle) ;
- La norme définit deux modèles de qualité : le modèle se référant à la qualité en production et celui mesurant la qualité à l'utilisation ;
- La sécurité est devenue une caractéristique au lieu d'être une sous-caractéristique de la fonctionnalité ;
- La portabilité a été scindée en deux : d'un côté la portabilité qui fait référence uniquement à la facilité d'installation et l'interchangeabilité, et de l'autre la compatibilité qui inclut l'interopérabilité ;
- Les sous-caractéristiques suivantes ont été ajoutées : robustesse, utilité, accessibilité technique, modularité, réutilisabilité et portabilité ;
- Plusieurs caractéristiques et sous-caractéristiques ont été précisées et renommées.

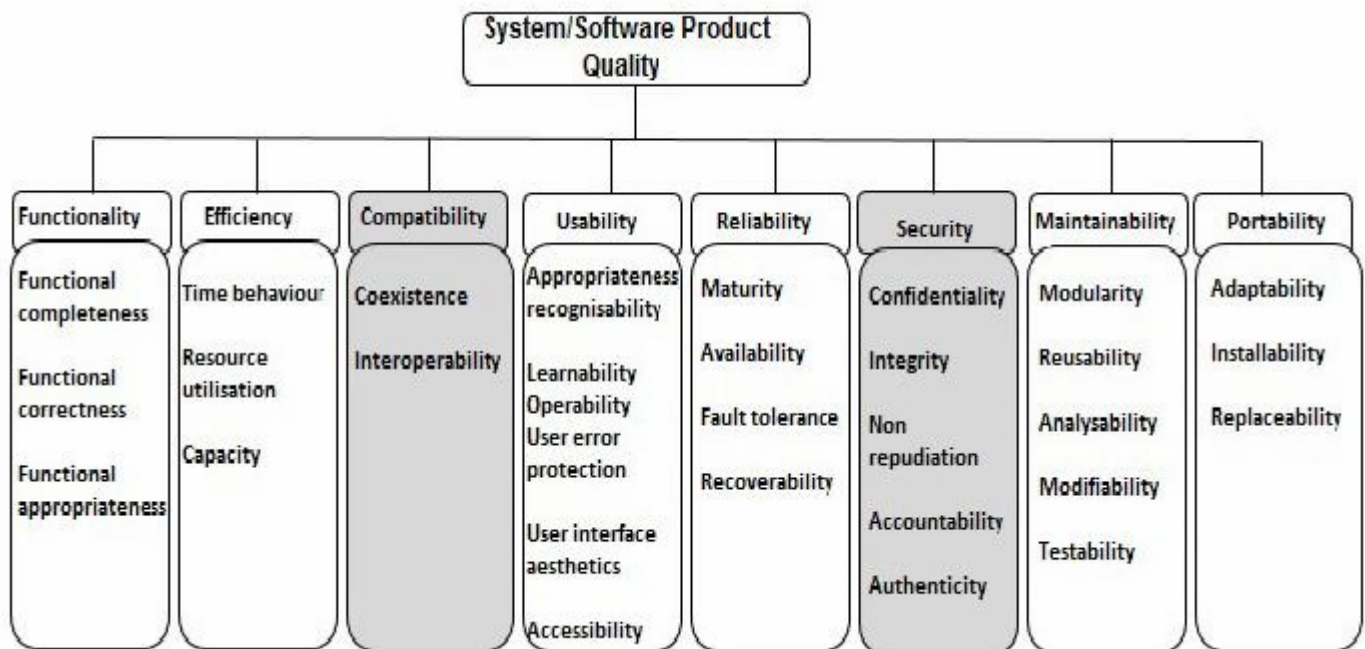


FIGURE 1.5 – Modèle SQuaRE

1.1.3.5 Dromey

Dans son travail [Dro95], Dromey souligne que les logiciels ne manifestent pas directement des attributs de qualité de haut niveau. Les logiciels ne possèdent que des caractéristiques de produit qui influencent les attributs de qualité. De mauvaises caractéristiques de produit réduisent ses attributs de qualité. Les modèles mentionnés ci-dessus n'établissent pas de lien explicite entre les attributs de qualité et les caractéristiques du produit. Le modèle de Dromey se concentre sur le produit logiciel principal, le code. Cette orientation vers le produit est la question la plus importante du travail de Dromey.

Selon Dromey, une décomposition directe des attributs dans le style du modèle ISO n'est pas la meilleure façon de procéder, car cela ne conduit qu'à d'autres attributs vagues. Il propose un seul niveau de "propriétés porteuses de qualité" entre les attributs de haut niveau et les composants du produit. C'est ce qui a donné naissance à son modèle générique (voir figure 1.6). Ce cadre permet une modélisation descendante (à chaque attribut de qualité de haut niveau, des propriétés porteuses de qualité peuvent être attribuées) ainsi qu'une modélisation ascendante (pour chaque composant, des propriétés porteuses de qualité sont identifiables, ce qui est important pour garantir les attributs de haut niveau).

Dans le contexte du développement de logiciels, Dromey met les "composants" au même niveau que les "formes structurelles" des langages de programmation (par exemple, les expressions, les variables, les boucles, etc.). L'ensemble des "formes structurelles" est déterminé par le langage de programmation. Dromey propose un ensemble de propriétés structurelles. Un aperçu est présenté dans le tableau 1.1. A titre d'exemple, nous discutons de la propriété porteuse de qualité "Assignée". Une variable est assignée si elle reçoit une valeur avant sa

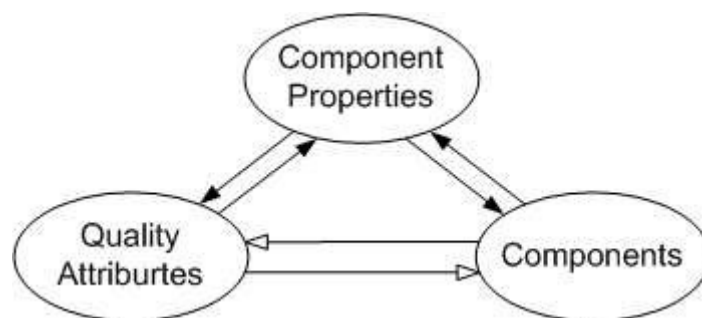


FIGURE 1.6 – Modèle générique de la Drôme [Dro95]. Le diagramme montre toutes les relations potentielles. Les flèches pleines sont importantes pour le modèle.

première utilisation. La propriété "assignée" peut donc être appliquée à des variables. Cela signifie que si notre code ne comporte pas de variables non assignées, nous avons intégré la qualité dans notre logiciel. Cette relation constitue la base du modèle de Dromey. Sur cette base, la connexion aux attributs de haut niveau peut être construite de manière similaire. Les propriétés du produit caractérisent les exigences qui doivent être satisfaites pour construire un attribut de qualité de haut niveau dans le logiciel. La partie la plus difficile dans cette situation est d'évaluer quelle propriété du produit a l'influence la plus significative sur l'attribut de qualité. Pour son modèle, Dromey a choisi une liste d'attributs de qualité, qui est similaire à la norme ISO 9126 (fonctionnalité, fiabilité, convivialité, efficacité, maintenabilité, portabilité). Comme extension de la norme ISO 9126, il a ajouté la réutilisabilité des attributs, qu'il considère comme un sujet important.

Propriétés de Correction	Propriétés Structurelles	Propriétés de Modularité	Propriétés Descriptives
programmable	structuré	paramétré	spécifié
complet	résolu	couplage faible (loosely coupled)	documenté
assigné	homogène	encapsul,	auto-descriptif
précis	effectif	cohésif	
initialisé	non-redondant	générique	
progressif	direct	abstrait	
variant	ajustable		
consistant	intervalle-indépendant		
	utilisé		

TABLE 1.1 – Liste des propriétés porteuses de qualité. Chacune de ces propriétés peut être appliquée à une ou plusieurs formes structurelles (les expressions, les variables, les boucles), et affecte un ou plusieurs attributs de qualité

1.1.3.6 Le modèle GQM (Goal Question Metrics)

L'objectif principal d'un modèle est la représentation d'une réalité afin de l'étudier. Cela passe par la collecte de toute information permettant de décrypter le modèle. Par syllogisme un modèle de qualité devrait permettre de mesurer les caractéristiques, c'est-à-dire proposer des métriques ayant un rapport avec les caractéristiques. Le modèle GQM proposé par [Basili et al., 1994] est une approche basée sur le concept de métrique. Ils considèrent que la mesure de la qualité commence par la définition d'objectifs ou buts (goals) à atteindre. La deuxième étape consiste en la définition des questions dont le propos est d'explicitier la façon dont les buts seront atteints. La troisième étape définit les mesures associées aux questions et dont l'interprétation donne une réponse à ces questions. Le résultat de l'application (l'implémentation) de l'approche GQM est la spécification d'un système de mesure orienté vers un ensemble de règles pour l'interprétation des données mesurées. Le modèle GQM constitué est composé de trois niveaux hiérarchiques avec comme point de départ les objectifs à atteindre de la figure 1.7 :

- Le niveau conceptuel (Goal) : ce niveau définit les buts ou objectifs à atteindre. Un but relève de différents points de vues selon lesquels le logiciel est analysé et peut concerner les différents artefacts logiciels, le processus de développement ou les ressources mises en œuvre dans le cadre de ce processus (spécification, modélisation, conception, test, etc.).
- Le niveau opérationnel (Question) : il est formé de l'ensemble des questions associées à un objectif particulier.
- Le niveau quantitatif (Metric) : il est formé de l'ensemble de données associées à une question, elles peuvent être objectives ou subjectives.

Le modèle GQM est essentiellement orienté organisation et projet. Il a d'ailleurs été implémenté à la NASA.

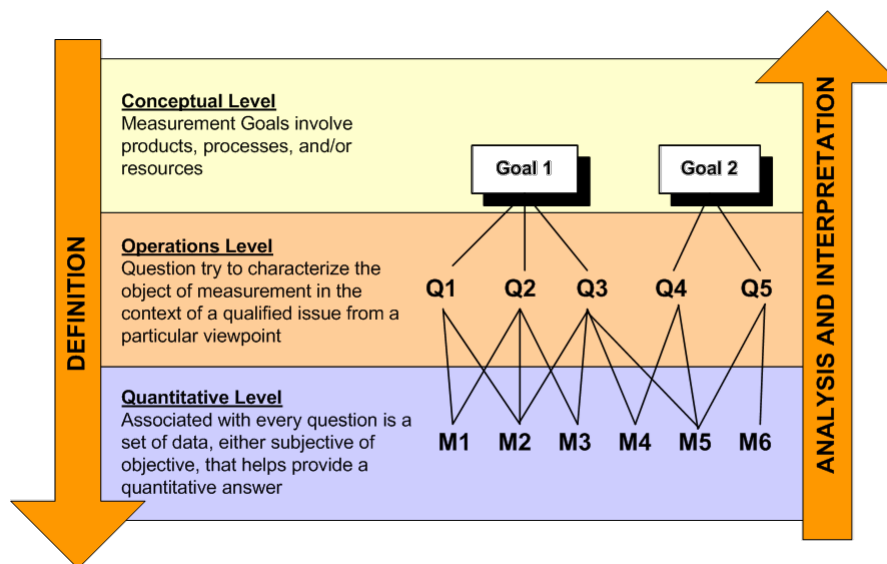


FIGURE 1.7 – Modèle GQM

1.2 Evaluation de la qualité Logicielle

Nous avons tout d'abord recensé les différents modèles de qualité les plus utilisés. La plupart de ces modèles permettent de déterminer la qualité d'un logiciel dans son ensemble et fournir une vue globale satisfaisante, notamment les modèles ISO-9126, SQuaRE tandis que d'autre notamment le modèle QCM donne à la fois une vue globale et une vue détaillée de la qualité logicielle. Comment donner un sens aux mesures sous l'angle de la qualité ? Les modèles hiérarchiques attribuent des notes déterminant le niveau de qualité d'un logiciel à partir de deux types différents de mesures : les métriques et les données brutes.

1.2.1 Évaluation avec les métriques

Une métrique est un moyen permettant de connaître la distance entre deux points. Appliquée à la production du logiciel, une métrique est un indicateur d'avancement ou de qualité des développements logiciels. Les métriques sont considérées à juste titre comme des indicateurs de la qualité par Basili [Basili et al., 1996]. Une métrique peut être simple (le nombre de lignes d'un programme) ou complexe (le couplage entre classes), la métrique n'est pas un état mais juste un reflet, une vision de la réalité. Aussi pour qualifier faut-il réaliser plusieurs mesures avec des instruments différents. Avoir une bonne idée d'un état, nécessite de disposer de plusieurs métriques. En métrologie, on a le plus souvent affaire à des données physiques, or le logiciel est une donnée abstraite. La transposition à l'ingénierie du logiciel a été l'objet de nombreux travaux fondateurs d'un cadre pour la mesure dans du logiciel les auteurs [Melton et al., 1990, Fenton, 1994, Weyuker, 1988, Briand et al., 1996], [Zuse and Bollmann, 1989, Zuse and Bollmann-Sdorra, 1992] de ces travaux ont plaidé pour un cadre théorique de la mesure pour l'ingénierie des logiciels. Ces travaux théoriques ont été complétés par des études théoriques sur la validation des mesures [Briand et al., 1995, Kitchenham et al., 1995, Schneidewind, 1992, Shepperd and Ince, 1991, Henderson-Sellers, 1996]. Les premiers travaux sur la mesure des applications logicielles procédurales furent ceux de Halstead [Halstead et

al., 1977] et [McCall et al., 1977] et concernaient des logiciels écrits en langage procédural. Chidamber [Chidamber and Kemerer, 1994] propose des mesures pour le paradigme orienté objet, mesures qui seront très activement étudiées et validées par des nombreux travaux au regard du succès du paradigme objet [Briand et al., 1998, Briand et al., 1999, Basili et al., 1996, Zuse, 2013, e Abreu and Melo, 1996].

1.2.2 Les métriques de code conventionnelles

Cette catégorie regroupe les métriques basiques couramment utilisées pour évaluer certaines propriétés du code source. Ces métriques s'appliquent à tout type de langage - procédural ou objet par exemple.

1.2.2.1 Halstead

Halstead est le pionnier dans le domaine de la métrique en génie logiciel. Il est le premier à proposer quatre mesures ou métriques sur le logiciel. Il part du constat, que les composants de base d'un programme telles que les expressions peuvent se classer en opérandes et opérateurs. De ces observations, il en déduit quatre métriques primitives de la façon suivante :

- n_1 nombre unique d'opérateurs dans un programme,
- n_2 le nombre unique d'opérandes dans un programme,
- N_1 le nombre total d'opérateurs dans un programme,
- N_2 le nombre total d'opérandes dans le programme.

A partir de ces quatre métriques, il crée un ensemble de métriques pour évaluer des propriétés du programme ainsi :

- $n = n_1 + n_2$ le vocabulaire d'un programme
- $N = N_1 + N_2$ la taille d'un programme
- $V = N \log_2 n_1$ le volume d'un programme
- $\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$ la taille estimé d'un programme

Nous ne donnerons pas ici toutes les métriques issues des travaux [3]. Dans ces mesures, il n'est pas clairement dit quels attributs de la qualité on mesure. De même, Halstead ne propose pas une validation de ces mesures. Néanmoins des travaux par la suite ont intégré les métriques d'Halstead dans la mesure de la complexité d'un programme apportant la décision sur sa maintenabilité.

1.2.2.2 McCabe : Complexité cyclomatique

Nom Complexité cyclomatique

Acronyme V(G)

Références [McCabe, 1976]

Définition : La complexité cyclomatique est la plus connue des mesures de base. Elle a été développée par Thomas McCabe en 1976 [4]. Cette mesure provient de la théorie des graphes, c'est le nombre cyclomatique qui indique le nombre de régions dans un graphe. Telle qu'appliquée dans le génie logiciel, c'est le nombre de chemins linéairement indépendants que comprend un programme. Elle peut être utilisée pour indiquer l'effort requis pour tester un logiciel. Pour déterminer les chemins, le programme logiciel est représenté comme un

1.2. EVALUATION DE LA QUALITÉ LOGICIELLE

graphe fortement connexe avec une entrée et une sortie unique. La formule pour calculer la complexité cyclomatique est la suivante :

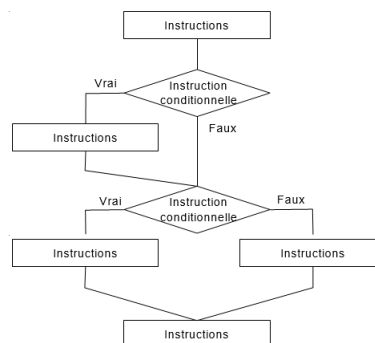
$M = V(G) = e - n + 2p$ où :

- $V(G)$ est le nombre cyclomatique du graphe G du programme,
- e est le nombre d'arcs,
- n est le nombre de nœuds,
- p est le nombre de composantes connexes du graphe. Généralement p est égal à 1.

La complexité cyclomatique est additive. La complexité de plusieurs graphes considérés comme un groupe égale la somme des complexités individuelles de chaque graphe. La complexité cyclomatique ignore la complexité des éléments séquentiels. Elle s'applique donc dans les programmes à des imbrications de boucles ou d'instructions de test. La mesure de la complexité cyclomatique se veut indépendante du langage : elle se calcule de la même façon pour tous les langages qu'ils soient fonctionnels, procéduraux ou orientés objets. La complexité cyclomatique est utilisée dans le cadre de la maintenabilité, de la testabilité et la compréhensibilité des programmes.

Dans cette formule, les arcs sont les branches d'un programme, les nœuds sont les blocs d'instructions séquentielles. De plus, tel qu'est défini le graphe de flot de contrôle, p a pour valeur 1 : il n'existe qu'un seul composant connecté. La figure 1.2 représente l'exemple suivant : le schéma supérieur représente le graphe du programme et les trois graphes inférieurs représentent chacun un chemin possible dans ce graphe. Dans cet exemple, e vaut 8, n vaut 7 et p vaut 1. Donc la valeur de $V(G)$ est de 3. Il y a effectivement 3 chemins linéairement indépendants dans cet exemple (ils sont représentés en couleur sur l'exemple). La complexité cyclomatique d'un programme a pour valeur minimum 1 puisqu'il y a toujours au moins un chemin possible.

Portée Fonction, Méthode, Classe



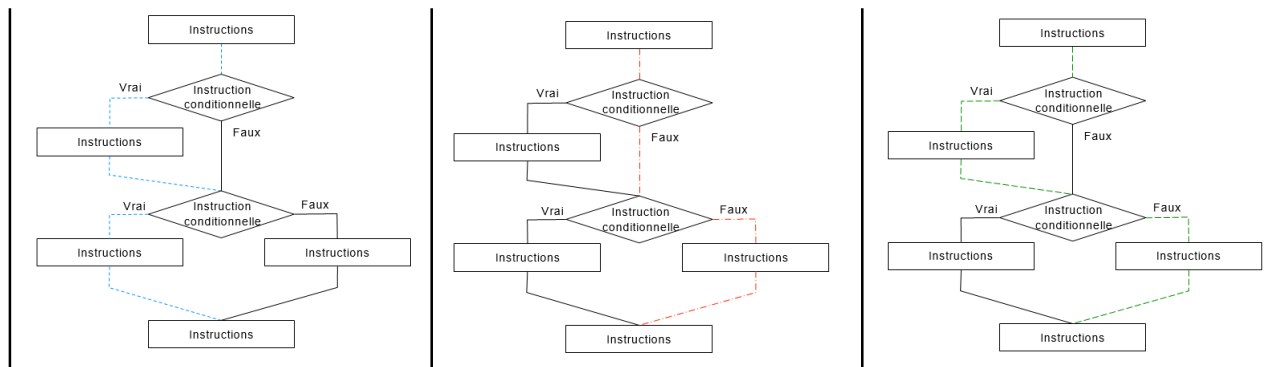


TABLE 1.2 – Un exemple de graphe de flot de contrôle avec les trois différents chemins linéairement indépendants.

1.2.2.3 Les lignes de code

Les mesures les plus largement utilisées concernent les lignes de code. Très simples à comprendre, ces métriques doivent pourtant être définies précisément. En effet, la composition d'une ligne de code varie et le moyen de compter les lignes de code n'est pas toujours compris de la même manière. Prenons l'exemple suivant, sans difficulté apparente : `for (i = 0 ; i < 10 ; i+ = 1) printf("hello World") ; /* une ligne de code avec une boucle */` Cet exemple contient une ligne de code mais plusieurs instructions et également un commentaire. On peut donc considérer que cette ligne contient :

- une ligne physique ;
- deux lignes logiques ;
- une ligne de commentaires.

Cet exemple illustre donc les différentes métriques que nous pouvons appliquer aux lignes de code.

SLOC

Nom : Nombre de lignes de code source

Acronyme : SLOC

Références : [Kan, 2002]

Définition La métrique SLOC pour Source Line Of Code mesure la taille d'un logiciel en déterminant le nombre de lignes de son code source. Cette métrique ne tient en principe pas compte des commentaires. Cependant, même si le plus courant est de compter le nombre de lignes physiques d'un programme sans tenir compte des lignes vides, certains utilitaires vont compter le nombre de lignes logiques ou encore vont avoir leur propre définition de dénombrement, comme l'illustre Park à travers la check-list qui définit les règles de calcul de SLOC [Park, 1992].

Portée : Fonction, Méthode

CLOC

Nom : Lignes de commentaires

Acronyme : CLOC

Références : [Kan, 2002]

Définition : La métrique CLOC pour Comments Lines Of Code détermine le nombre de lignes de commentaires dans un programme.

Portée ; Fonction, Méthode

1.2.2.4 Les métriques de code orienté objet

Les métriques conventionnelles sont de bons indicateurs mais ne correspondent pas toujours aux exigences et particularités du code orienté objet. En particulier Moreau et Dominick ont montré que les métriques traditionnelles ne s'appliquent pas au design orienté objet [Moreau et Dominick, 1989]. Par exemple, mesurer le nombre de lignes de code (SLOC) ne leur apparaît pas suffisant dans le cadre orienté objet. C'est pourquoi ils suggèrent d'associer cette mesure avec par exemple, la profondeur d'héritage pour prendre en compte la réelle complexité du code. Par ailleurs, Tegarden présente le résultat d'une série de mesures sur quatre systèmes et prouve que les métriques traditionnelles telles que SLOC, la complexité cyclomatique ou encore les métriques de Halstead sont utiles pour interpréter la complexité du design orienté objet mais que ces seules métriques ne suffisent pas [Tegarden et al., 1992] .

1.2.2.5 Les métriques de Chidamber et Kemerer

Les métriques suivantes, parfois appelées les métriques CK, ont été regroupées par Chidamber et Kemerer comme constituant un ensemble de métriques de base d'analyse du code source orienté objet [Chidamber et Kemerer, 1994] :

WMC pour Weighted Methods per Class ;

DIT pour Depth of Inheritance Tree ;

NOC pour Number Of Children ;

RFC pour Response For a Class ;

LCOM pour Lack of Cohesion in Methods ;

CBO pour Coupling Between Objects

Nom Nombre pondéré de méthodes par classe (Weighted Methods per Class)

Acronyme : WMC

Références : [Chidamber et Kemerer, 1994]

Définition WMC comptabilise le nombre de méthodes définies pour une classe donnée, pondérées par leur complexité, sachant que la complexité s'entend ici en terme de complexité cyclomatique.

Portée : Classe

Nom Profondeur de l'arbre d'héritage (Depth of Inheritance Tree)

Acronyme : DIT

Références : [Lorenz et Kidd, 1994],[Chidamber et Kemerer, 1994],[Briand et al., 1996],[Gyimóthy et al., 2005],[Li et Henry, 1993],[Hitz et Montazeri, 1995],[Tempero et al., 2008]

Définition La profondeur d'héritage d'une classe correspond à la profondeur maximum de la classe dans l'arbre d'héritage, i.e., depuis la racine jusqu'au nœud de la classe, mesurée en nombre d'ancêtres de la classe. Plus une classe est profonde dans l'arbre d'héritage, plus le nombre de méthodes héritées est élevé, ce qui rend la prédiction et la compréhension de son comportement plus complexe. Un arbre d'héritage profond résulte d'un design complexe

avec des classes et des méthodes très imbriquées. En revanche, cela diminue le potentiel de réutilisation des méthodes héritées.

Portée : Classe

Nom Nombre de fils dans l'arbre d'héritage (Number Of Children)

Acronyme : NOC

Références : [Chidamber et Kemerer, 1994, Gyimóthy et al., 2005]

Définition Comptabilise le nombre de sous-classes immédiatement subordonnées à la classe dans la hiérarchie.

Portée : Classe

Nom Réponse pour une classe (Response For a Class)

Acronyme : RFC

Références : [Chidamber et Kemerer, 1994]

Définition RFC compte le nombre de méthodes accessibles par la classe, qu'elles soient implémentées directement, surchargées ou disponibles par héritage (méthodes publiques, protégées et du package, pour Java). Elle correspond à la somme du nombre de méthodes héritées (NIM - voir plus loin), du nombre de méthodes surchargées (NRM - voir plus loin) et du nombre de méthodes implémentées (WMC).

Portée : Classe

1.2.2.6 Les métriques de Lorenz et Kidd

Lorenz et Kidd ont déterminé également un certain nombre d'autres métriques de base dont nous retiendrons les principales [Lorenz et Kidd, 1994] :

NOM pour Number Of Methods ;

NIM pour Number of Inherited Methods ;

NRM pour Number of overRiden Methods ;

SIX pour Specialization IndeX.

Nous donnons maintenant une définition précise de chacune de ces métriques.

Nom Nombre de méthodes (Number of Methods)

Acronyme :NOM

Références [Lorenz et Kidd, 1994]

Définition NOM comptabilise le nombre de méthodes définies localement à une classe, comptant les méthodes privées aussi bien que publiques. Les méthodes surchargées sont également comptabilisées.

Portée ; Classe

Nom Nombre de méthodes héritées (Number of Inherited Methods)

Acronyme : NIM

Références [Lorenz et Kidd, 1994, Briand et al., 1998a]

Définition NIM est une métrique simple qui permet de mesurer le nombre de comportements/propriétés qu'une classe peut réutiliser. Elle compte le nombre de méthodes héritées par une classe : les méthodes auxquelles une classe peut accéder par l'intermédiaire de ses super-classes.

Portée : Classe

Nom Nombre de méthodes surchargées (Number of overRiden Methods)

Acronyme : NRM

Références : [Lorenz et Kidd, 1994]

Définition NRM comptabilise le nombre de méthodes surchargées i.e., définies dans la super-classe et redéfinies dans la classe. Cette métrique inclut les méthodes super.

Portée : Classe

Nom Index de spécialisation (Specialization Index)

Acronyme : SIX

Références [Lorenz et Kidd, 1994, Mayer, 1999]

Définition

$$SIX = \frac{NRM * DIT}{NOM + NIM}$$

Cette métrique détermine le niveau de spécialisation d'une classe en tenant compte de sa profondeur d'héritage. Cette métrique fait le ratio entre le nombre de méthodes surchargées et le total des méthodes définies dans la classe, pondéré par la profondeur d'héritage. Lorenz et Kidd précisent que les méthodes qui invoquent les méthodes super ou qui surchargent des *templates* ne sont pas incluses. La redéfinition et la surcharge de méthodes sont de moins en moins souhaitables au fur et à mesure où l'on descend dans la hiérarchie d'une classe. Cela augmente la complexité du développement. **Portée :** Classe

1.2.2.7 Les métriques de design

Ces métriques évaluent les entités d'un code source par rapport au respect des principes de design. Ces métriques, en faisant ressortir les entités qui ne respectent pas ces principes, mettent en évidence les lacunes générales du design dont la correction peut amener une amélioration globale de la qualité. Nous distinguons les métriques qui traitent du couplage, les métriques qui mesurent la cohésion de classes et les métriques qui évaluent la cohésion des packages.

Nom Couplage entre les objets (Coupling Between Object classes)

Acronyme : CBO

Références : [Chidamber et Kemerer, 1994],[Fenton et Pfleeger, 1996],[Martin, 2005]

Définition : Deux classes sont couplées ensemble si l'une d'elles utilise l'autre, i.e., une classe appelle une méthode ou accède à un attribut d'une autre classe. Le couplage à travers l'héritage et le polymorphisme sont également pris en compte. Pour une classe, la métrique CBO compte le nombre de classes auxquelles elle est reliée, couplée.

Portée : Classe

Nom Manque de cohésion des méthodes (Lack of COhesion in Methods)

Acronyme : LCOM1

Références : [Chidamber et Kemerer, 1994][Briand et al., 1998b]

Définition LCOM1 est le nombre de paires de méthodes dans une classe qui ne référencent pas d'attribut commun.

Portée : Classe

Envisageons une évaluation avec les métriques. Il se pose deux principaux problèmes. Tout

d'abord, les métriques sont définies et mesurées pour certains composants du logiciel : la métrique SLOC (nombre de lignes de code source) par exemple est calculée pour une méthode donnée ou encore la métrique DIT (profondeur d'héritage) calculée pour une classe donnée. Il n'est donc pas aisé de transposer les résultats obtenus pour des composants vers une évaluation qualitative de plus haut niveau. D'autre part, les principes qualitatifs tels qu'ils sont définis font le plus souvent appel à l'utilisation de plusieurs métriques. Par exemple, la norme ISO 9126 définit la sous-caractéristique facilité de modification comme "la capacité d'un logiciel à intégrer de nouvelles implémentation". Pour mesurer cette propriété, les métriques telles que le nombre de lignes de code (SLOC), la complexité cyclomatique, le nombre de méthodes par classe, la profondeur d'héritage (DIT) sont combinées de manière à déterminer à partir de toutes ces mesures une seule et unique note pour cette sous-caractéristique.

Pour parvenir à évaluer un critère de qualité à partir de métriques il faut donc procéder en deux étapes : la combinaison. La première consiste à combiner les métriques retenues entre-elles. Cette étape qui pourra être exécutée de différentes manières permet d'obtenir une note qualitative pour un composant donné : par exemple, en combinant la métrique CLOC avec la métrique V(G) pour chaque méthode d'un projet. Nous nommons cette étape la combinaison. On cherche en effet à donner du sens, un sens qualitatif en combinant des mesures différentes ; l'agrégation. La seconde étape consiste à agréger ces notes obtenues au niveau de chacun des composants en une note globale pour la totalité du projet. Nous nommons cette étape l'agrégation. Celle-ci relève plus d'un aspect statistique. On détermine à partir d'un ensemble de mesures la valeur générale qui s'en dégage.

La combinaison de métriques

L'étape de combinaison des métriques implique de tenir compte des intervalles de mesure de celle-ci, ce qui entraîne deux points importants. Tout d'abord, les intervalles des métriques peuvent être totalement différents, par exemple, le critère facilité de modification qui utilise les métriques SLOC, V(G), et DIT. Ces trois métriques ne possèdent pas les mêmes échelles de valeurs. Il faut alors s'assurer que la combinaison ne dilue pas les mesures de l'une par rapport à l'autre. Ensuite, les métriques possèdent toutes leur propre sens : SLOC renseigne sur le nombre de lignes d'une méthode, tandis que V(G) nous fournit une information sur la complexité, et DIT qualifie la profondeur d'héritage. Ceci impose de les utiliser de manière différente et de faire en sorte de garder le sens des unes par rapport aux autres. Pour y parvenir, la combinaison des métriques doit être élaborée spécifiquement pour chaque critère évalué. Dans la table 3.1 à propos du taux de commentaires d'un code, par exemple, l'opération de multiplication de CLOC avec V(G) n'est pas satisfaisante. Il serait préférable de combiner ces deux métriques plus finement pour traduire le fait que le commentaire doit être mis en perspective avec la complexité, comparer la complexité avec les commentaires sous forme d'un seuil, par exemple.

L'Agrégation des métriques

L'étape d'agrégation des mesures doit elle aussi être effectuée de manière à ne pas perdre les informations fournies au niveau de chaque composant. Comment faire ressortir le fait qu'un élément ne réponde pas aux exigences de qualité lorsqu'on se situe au niveau le plus haut ? Utiliser des notes globales pour définir la qualité pose également un problème crucial pour les développeurs : comment retrouver les informations livrées par les données brutes

à travers une seule note globale ? Comment traduire cette note en un problème concret de conception/développement ? Cet écueil empêche nombre de développeurs de s'intéresser à un modèle de qualité dans son ensemble et ils lui préfèrent encore souvent les métriques de code brutes. Pour fournir une représentation de la qualité à un niveau élevé, le modèle ISO 9126 s'appuie sur des métriques, comme décrit dans sa plus haut[6]. Cependant, cette description ne donne aucune indication précise quant à la manière d'agréger les différentes métriques citées. Une moyenne simple ou pondérée reste souvent le moyen le plus utilisé pour y parvenir. Et pourtant, nous allons voir que les moyennes ne donnent pas entière satisfaction puisqu'elles perdent de l'information comme cela est souligné par Bieman et d'autres chercheurs [7, 8, 9]

Moyenne simple

La méthode employée pour calculer une note globale sans perdre les informations fournies par les notes individuelles des composants du projet cristallise souvent les points faibles des modèles

Methodes	SLOC Classe 1	SLOC Classe 2
A	24	71
B	25	9
C	27	10
D	24	8
Moyenne	25.0	24.5

TABLE 1.3 – Les moyennes simples de la métrique SLOC pour quatre méthodes dans deux classes différentes

qualité. Calculer une simple moyenne n'est pas assez précis puisque cela ne permet pas de déterminer l'écart type d'une population comme illustré ensuite. Elle dilue les valeurs extrêmes au milieu des valeurs moyennes. La Table 1.3 présente le nombre de lignes de code, soit la métrique SLOC, pour quatre méthodes dans deux classes différentes. Dans cet exemple, la moyenne du nombre de lignes de code est de 25.0 pour la classe 1 et de 24.5 pour la classe 2. En partant du fait qu'il est préférable pour une classe de posséder des méthodes n'ayant pas un nombre trop élevé de lignes de code, ces résultats pourraient amener à croire que la seconde classe est de meilleure qualité que la première (puisque la moyenne est moins élevée) ou du moins que ces deux classes sont sensiblement identiques. Mais cette moyenne masque le fait que la seconde classe possède une méthode A qui est très clairement en dehors des normes. C'est pourquoi, bien que la note moyenne soit meilleure, le détail des notes montre que cette seconde classe est pourtant la moins bonne, du moins par rapport aux mesures de la métrique SLOC. La moyenne, parce qu'elle lisse les résultats ne représente pas toujours la réalité [9]. Pour être utile, un modèle de qualité doit être un modèle d'évaluation mais également un guide pour augmenter la qualité. Un développeur doit pouvoir connaître les composants à améliorer et un manager les points faibles du projet : donner une note globale qui ait du sens, mettre en avant les mauvais composants et les faiblesses d'une application. Dans l'exemple précédent, un indicateur de qualité approprié devrait pointer du doigt le mauvais résultat de la méthode A en fournissant une note globale plus basse. Une simple

1.2. EVALUATION DE LA QUALITÉ LOGICIELLE

moyenne ne pointe pas les mauvais composants et même pire : elle les masque. En effet, si l'on prend pour exemple la règle qualitative suivante : "les méthodes de plus de 300 lignes de code sont inacceptable", une moyenne simple peut facilement échouer à traduire cette règle pour les raisons susdites. Pour remédier à cet inconvénient, il est souvent décidé d'utiliser une moyenne pondérée. Cependant, cette méthode a aussi ses défauts comme nous en discutons dans la suite.

Moyenne pondérée

L'idée principale derrière l'utilisation d'une moyenne pondérée est de mettre en avant les mauvais composants et de détecter s'il existe des composants critiques. Intuitivement, il s'agit d'utiliser l'agrégation de métriques comme une alarme : donner une mauvaise note globale lorsqu'un composant est mauvais. Le poids est appliqué aux notes individuelles et représente l'influence de la note comparée aux autres.

Note	≤ 35	$]35;70]$	$]70;160]$	> 160
Poids	1	3	9	27

TABLE 1.4 – Exemple de poids appliqués sur SLOC

Méthodes	SLOC	Poids	SLOC pondérée	Méthodes	SLOC	Poids	SLOC pondérée
A	30	1	30	A	25	1	25
B	50	3	150	B	30	1	30
C	70	9	630	C	50	3	150
D	300	27	8100	D	300	27	8100
Moyenne Simple/Pondéré	112.5		222.75	Moyenne Simple/Pondéré	101.25		259.53

TABLE 1.5 – Les moyennes de deux versions différentes d'un même projet

Considérons l'exemple suivant : la Table 1.4 décrit les poids donnés pour la métrique SLOC dans la première version de Squale. Ces poids ont été choisis pour traduire l'intention sous-jacente de pondérer de plus en plus sévèrement les mauvais résultats. Les poids sont multipliés par 3 pour chaque seuil et les valeurs des seuils de la métrique SLOC sont déterminées par les développeurs en fonction de leur savoir-faire :

- une méthode de moins de 35 lignes est idéale ;
- une méthode entre 35 et 70 lignes est acceptable ;
- une méthode entre 70 et 160 lignes est passable ;
- une méthode de plus de 160 lignes est inacceptable.

Les mesures obtenues pour cette métrique sont donc pondérées de plus en plus fortement, avec une valeur extrême de 27, pour augmenter leur influence lors du calcul de la moyenne. La Table 1.5 représente un exemple de résultats obtenus pour la métrique SLOC selon ce principe pour deux versions d'un même projet. Par exemple, les poids appliqués pour la

méthode C sont différents du fait de la note différente obtenue. Les méthodes B et C illustrent le paradoxe créé par l'emploi de cette technique. En effet, alors que la valeur de la métrique SLOC diminue et donc que la qualité augmente, les poids appliqués aux valeurs produisent un effet totalement inverse sur le calcul de la note globale : celle-ci diminue ! Dans cet exemple, la moyenne pondérée passe de $(30 + 150 + 630 + 8100) / 40 = 222,75$ à $(25 + 30 + 150 + 8100) / 32 = 259,53$ pour la seconde version car la somme des poids passe de 40 à 32. Le résultat augmente (donc l'évaluation de la qualité diminue) alors même que la qualité du code est globalement améliorée. Cet exemple montre que l'utilisation d'une moyenne pondérée n'est pas la méthode adéquate et peut même s'avérer totalement inappropriée. Un modèle de qualité doit refléter tous les changements le plus finement possible et avec fiabilité.

Conclusion

Dans cette partie nous venons définir ce qu'est la qualité plus particulièrement la qualité logicielle, ensuite nous avons donné les différents modèles de la qualité existant dans la littérature et leurs différentes limites et enfin l'évaluation de la qualité logicielle à partir des métriques. Dans ce qui suit nous définirons le Machine Learning, donnerons l'historique de ce dernier et présenterons les différents types de Machine Learning et les différents algorithmes de Machine Learning existants et enfin son apport pour l'évaluation de la qualité logicielle.

1.3 Définition et Historique du Machine Learning

1.3.1 Définition

Le Machine Learning est une discipline de l'Intelligence Artificielle qui offre aux ordinateurs la possibilité d'apprendre à partir d'un ensemble d'observations que l'on appelle ensemble d'apprentissage. Chaque observation, comme par exemple « *j'ai mangé tels et tels aliments à tel moment de la journée pendant telle période ce qui a causé telle maladie* » est décrite au moyen de deux types de variables :

- Les premières sont appelées les variables prédictives (ou attributs ou caractéristiques), dans notre exemple mon âge, mon dossier médical, mes antécédents médicaux. Ce sont les variables à partir desquelles on espère pouvoir faire des prédictions. Les n variables prédictives associées à une observation seront notées comme un vecteur $x = (x_1, \dots, x_n)$ à n composantes. Un ensemble de M observations sera constitué de M tels vecteurs $x^{(1)}, \dots, x^{(M)}$.
- Une variable cible dont on souhaite prédire la valeur pour des événements non encore observés. Dans notre exemple, il s'agirait de la maladie contractée. On notera y cette variable cible.

En résumé, la valeur de la variable y dépend de :

- Une fonction $F(x)$ déterminée par les variables prédictives.

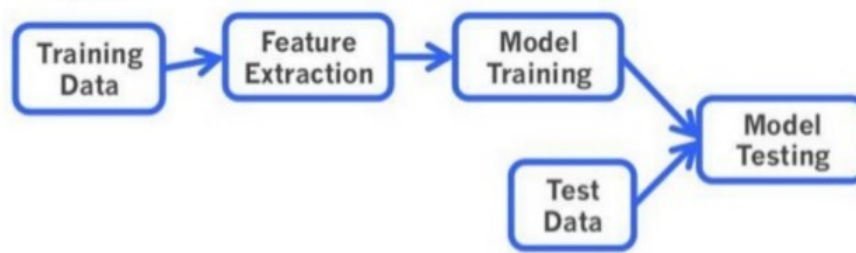


FIGURE 1.8 – le processus typique du Machine Learning

- Un bruit $\epsilon(x)$ qui est le résultat d'un nombre de paramètres dont on ne peut pas tenir compte.

Aussi bien F que ϵ ne seront jamais connues mais l'objectif d'un modèle de Machine Learning est d'obtenir la meilleure approximation possible de F à partir des observations disponibles. Cette approximation sera notée f , on l'appelle la fonction de prédiction.

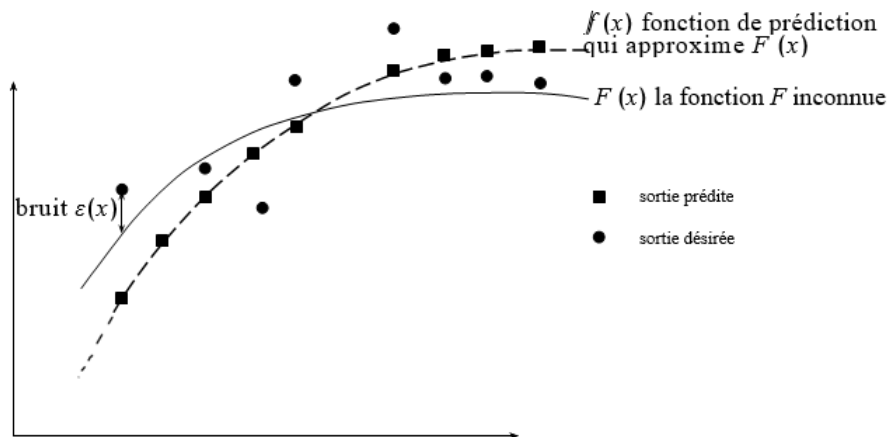


Figure 2 - un modèle de Machine Learning qui essaye d'obtenir la meilleure approximation possible de F

Voici quelques exemples d'utilisation du Machine Learning :

- Vision par ordinateur [10]
- Détection de fraude [11]
- Classification (image, texte, video, son, ...)
- Les publicités ciblées [12]
- Diagnostic médical [13]

1.3.2 Historique du Machine Learning

Année	Contributeur	Contribution
1943	Warren McCulloch et Walter Pitts	Introduction du McCulloch-Pitts (MCP) modèle considéré comme l'ancêtre des réseaux de neurones artificiels

1.3. DÉFINITION ET HISTORIQUE DU MACHINE LEARNING

Année	Contributeur	Contribution
1950	Alan Turing	Alan Turing crée le «test de Turing » pour déterminer si un ordinateur dispose d'une véritable intelligence. Pour réussir le test, un ordinateur doit être capable de tromper un humain en lui faisant croire qu'il est aussi humain.
1952	Arthur Samuel	Arthur Samuel a écrit le premier programme d'apprentissage informatique. Le programme était le jeu de dames, et l'ordinateur d'IBM s'améliorait au fur et à mesure qu'il jouait, en étudiant quelles techniques constituaient des stratégies gagnantes et en intégrant ces stratégies à son programme.
1957	Frank Rosenblatt	Frank Rosenblatt conçoit le premier réseau de neurones pour ordinateurs (le perceptron), qui simule les processus de pensée du cerveau humain. L'objectif principal de ceci était la reconnaissance des formes et des motifs.
1967	*	L'algorithme du « plus proche voisin » est écrit, permettant aux ordinateurs de commencer à utiliser une reconnaissance de motif très basique. Cela pourrait être utilisé pour tracer un itinéraire pour les vendeurs qui voyagent, en partant d'une ville au hasard, mais en s'assurant qu'ils visitent toutes les villes au cours d'une courte visite.
1979	Les étudiants de l'Université de Stanford	Inventent le « <i>Stanford Cart</i> », qui peut franchir seul les obstacles dans une pièce.
1981	Gerald Dejong	Introduit le concept d'apprentissage basé sur l'explication (EBL), dans lequel un ordinateur analyse les données de formation et crée une règle générale qu'il peut suivre en éliminant les données sans importance.
1985	Terry Sejnowski	Terry Sejnowski Invente NetTalk qui apprend à prononcer des mots comme un bébé.
Années 1990	*	Le travail sur l'apprentissage automatique passe d'une approche basée sur la connaissance à une approche basée sur les données. Les scientifiques commencent à créer des programmes informatiques pour analyser de grandes quantités de données et tirer des conclusions - ou « apprendre » - à partir des résultats.
1997	IBM	Deep Blue d'IBM bat le champion du monde aux échecs.

1.3. DÉFINITION ET HISTORIQUE DU MACHINE LEARNING

Année	Contributeur	Contribution
2006	Geoffrey Hinton	Invente le terme « <i>apprentissage en profondeur</i> » pour expliquer les nouveaux algorithmes permettant aux ordinateurs de « voir » et de distinguer les objets et le texte dans les images et les vidéos.
2010	Microsoft	Le Microsoft Kinect peut suivre 20 entités humaines à une vitesse de 30 fois par seconde, permettant aux utilisateurs d'interagir avec l'ordinateur par le biais de mouvements et de gestes.
2011	IBM	Le Watson d'IBM bat ses concurrents humains chez Jeopardy.
2011	GOOGLE	Le cerveau est développé et son réseau de neurones profonds peut apprendre à découvrir et à classer des objets de la même manière qu'un chat.
2012	GOOGLE	Un laboratoire de Google développe un algorithme d'apprentissage automatique capable de parcourir des vidéos YouTube de manière autonome pour identifier celles contenant des chats.
2014	FACEBOOK	Facebook développe DeepFace, un algorithme logiciel capable de reconnaître ou de vérifier des personnes sur des photos au même niveau que l'être humain.
2015	AMAZON	Amazon lance sa propre plate-forme d'apprentissage automatique.
2015	Microsoft	Microsoft crée la boîte à outils Distributed Machine Learning, qui permet de distribuer efficacement les problèmes d'apprentissage machine sur plusieurs ordinateurs.
2015	*	Plus de 3 000 chercheurs en intelligence artificielle et en robotique, appuyés par Stephen Hawking, Elon Musk et Steve Wozniak (parmi beaucoup d'autres), signent une lettre ouverte mettant en garde contre le danger des armes autonomes sélectionnant et engageant des cibles sans intervention humaine.
2016	GOOGLE	L'algorithme d'intelligence artificielle de Google bat un joueur professionnel au jeu de société chinois Go, considéré comme le jeu de société le plus complexe au monde et beaucoup plus dur que les échecs. L'algorithme AlphaGo développé par Google DeepMind a réussi à gagner cinq jeux sur cinq dans la compétition Go.

TABLE 1.6 – Historique

[14]

1.3.3 Performance et sur-apprentissage

On peut penser que la performance d'un modèle sera en fonction des prédictions correctes faites sur l'ensemble d'observation utilisées pour l'apprentissage, plus elle est élevée, mieux c'est. Pourtant c'est complètement faux, ce qu'on cherche à obtenir du Machine Learning n'est pas de prédire avec exactitude les valeurs des variables cibles connues puisqu'elles ont été utilisées pour l'apprentissage mais bien de prédire celles qui n'ont pas encore été observées. Par conséquent, la qualité d'un algorithme de Machine Learning se juge sur sa capacité à faire les bonnes prédictions sur les nouvelles observations grâce aux caractéristiques apprises lors de la phase d'entraînement. Il faut donc éviter le cas où on a un modèle de Machine Learning tellement trop entraîné qu'il arrive à prédire à la perfection les données d'apprentissage mais qui n'arrive pas à généraliser sur les données de test. On l'appelle le sur-apprentissage. La cause du sur-apprentissage est que le modèle est trop complexe par rapport à la fonction F que l'on souhaite apprendre.

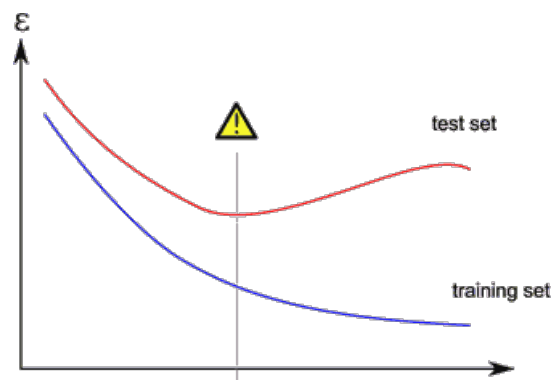


FIGURE 1.9 – Le sur-apprentissage : le graphe montre l'évolution de l'erreur commise sur l'ensemble de test par rapport à celle commise sur l'ensemble d'apprentissage, les deux erreurs diminuent mais dès que l'on rentre dans une phase de sur-apprentissage, l'erreur d'apprentissage continue de diminuer alors que celle du test augmente.

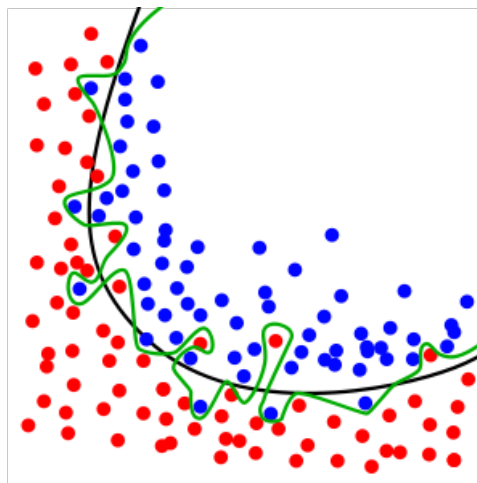


FIGURE 1.10 – La ligne verte représente un modèle surentraîné et la ligne noire représente un modèle régularisé. Ce dernier aura une erreur de test moins importante.

Pour résoudre ce problème, on divise les données disponibles en deux groupes distincts. Le premier sera l'ensemble d'apprentissage, et le deuxième sera l'ensemble de test.

Pour avoir une bonne séparation des données en données d'apprentissage et données de test, on utilise la validation croisée. L'idée c'est de séparer aléatoirement les données dont on dispose en k parties séparées de même taille. Parmi ces k parties, une fera office d'ensemble de test et les autres constitueront l'ensemble d'apprentissage. Après que chaque échantillon ait été utilisé une fois comme ensemble de test. On calcule la moyenne des k erreurs moyennes pour estimer l'erreur de prédiction.

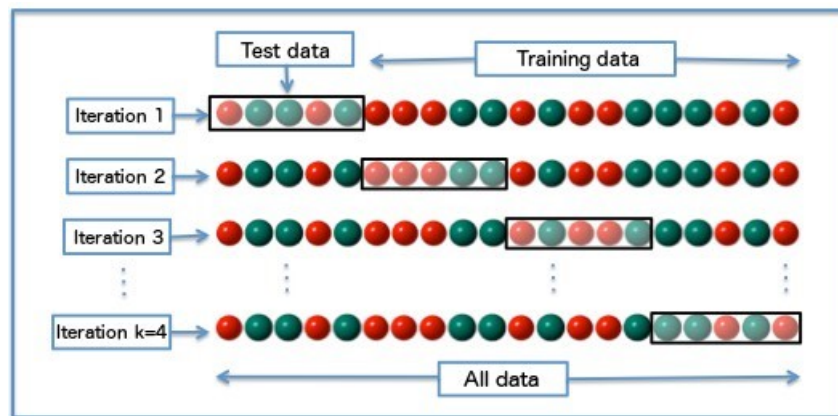


FIGURE 1.11 – La validation croisée.

1.3.4 Les différents types de machine Learning

1.3.4.1 Apprentissage supervisé et non supervisé

- La tâche de l'apprentissage supervisé c'est : soit l'ensemble d'apprentissage composé de N exemples de paire entrée-sortie : $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(M)}, y^{(M)})$ chaque $y^{(j)}$ a été généré par une fonction $F(x) = y$ inconnue, découvrir la fonction f qui se rapproche de F
- L'apprentissage non supervisé ou clustering ne demande aucun étiquetage préalable des données. Le but est que le modèle réussisse à regrouper les observations disponibles en catégories par lui-même l'apprentissage semi supervisé est à mi chemin entre ces deux méthodes. On fournit au modèle quelques exemples étiquetés mais la grande partie des données ne le sont pas. On trouve des cas d'application partout où l'obtention des données est facile mais leur étiquetage demande des efforts, du temps ou de l'argent comme par exemple :
 - En reconnaissance de parole, il ne coûte rien d'enregistrer une grande quantité de parole, mais leur étiquetage nécessite des personnes qui les écoutent.
 - Des milliards de pages web sont disponibles, mais pour les classer il faut les lire.

1.3.4.2 Régression et Classification

- Un modèle de classification est un modèle de Machine Learning dont les sorties y appartiennent à un ensemble fini de valeurs (exemple : bon, moyen, mauvais)
- Un modèle de régression est un modèle de Machine Learning dont les sorties y sont des nombres (exemple : la température de demain)

1.4 Les différents type d'algorithme de Machine Learning

1.4.1 Le classifieur naïf de Bayes

Le classifieur naïf de Bayes est un algorithme supervisé probabiliste qui suppose que l'existence d'une caractéristique pour une classe, est indépendante de l'existence d'autres caractéristiques, raison pour laquelle on utilise l'adjectif «naïf». Une personne peut être considérée comme un homme si il pèse un certain poids et mesure une certaine taille. Même si ces caractéristiques sont liées dans la réalité, un classifieur bayésien naïf déterminera que la personne est un homme en considérant indépendamment ces caractéristiques de taille et de poids.

Malgré des hypothèses de base extrêmement simplistes, ce classifieur conduit à de très bons résultats dans beaucoup de situations réelles complexes. En 2004, un article a montré qu'il existe des raisons théoriques derrière cette efficacité inattendue [15]. Toutefois, une autre étude de 2006 montre que des approches plus récentes (arbres renforcés, forêts aléatoires) permettent d'obtenir de meilleurs résultats[16].

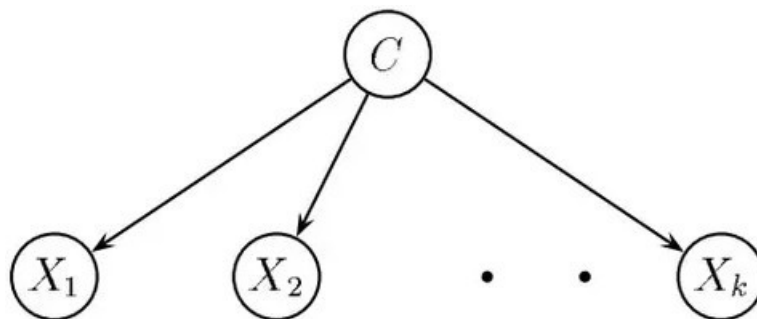


FIGURE 1.12 – Le classifieur naïf de Bayes est basé sur le théorème de Bayes avec une indépendance (dite naïve) des variables prédictives.

- Avantages :
 - L'algorithme offre de bonne performance
- Inconvénients
 - La prédiction devient erronée si L'Hypothèse indépendance conditionnelle est invalide

1.4.2 Les k plus proches voisins

L'algorithme des K-Nearest Neighbors (KNN) (K plus proches voisins) est un algorithme de classification supervisé. Chaque observation de l'ensemble d'apprentissage est représentée par un point dans un espace à n dimensions ou n est le nombre de variables prédictives. Pour prédire la classe d'une observation, on cherche les k points les plus proches de cet exemple. La classe de la variable cible, est celle qui est la plus représentée parmi les k plus proches voisins. Il existe des variantes de l'algorithme ou on pondère les k observations en fonction de leur distance à l'exemple dont on veut classer[17], les observations les plus éloignées de notre exemple seront considérées comme moins importantes.

Une variante de l'algorithme est utilisée par NetFlix [18] pour prédire les scores qu'un utilisateur attribuera à un film en fonction des scores qu'il a attribués à des films similaires

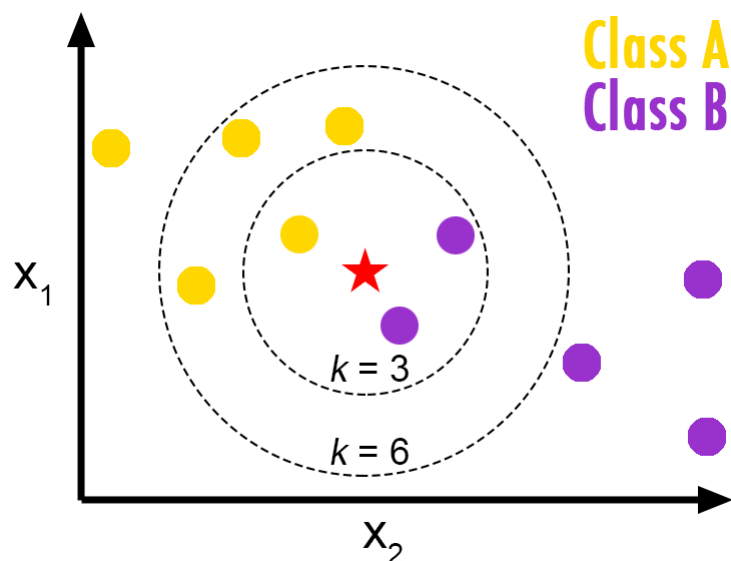


FIGURE 1.13 – Pour $k = 3$ la classe majoritaire du point central est la classe B, mais si on change la valeur du voisinage $k = 6$ la classe majoritaire devient la classe A

- Avantages :
 - Simple à concevoir
- Inconvénients
 - Sensible aux bruits
 - Pour un nombre de variable prédictives très grands, le calcul de la distance devient très coûteux.

1.4.3 Les arbres de décision

Les arbres de décision sont des modèles de Machine Learning supervisés, pouvant être utilisés pour la classification que pour la régression.

Un arbre de décision représente une fonction qui prend comme entrée un vecteur d'attributs et retourne une décision qui est une valeur unique. Les entrées et les sorties peuvent être

discrètes ou continues.

Un arbre de décision prend ses décisions en exécutant une séquence de test, chaque nœud interne de l'arbre correspond à un test de la valeur d'un attribut et les branches qui sortent du nœud sont les valeurs possibles de l'attribut. La classe de la variable cible est alors déterminée par la feuille dans laquelle parvient l'observation à l'issue de la séquence de test.

La phase d'apprentissage consiste à trouver la bonne séquence de test. Pour cela, on doit décider des bons attributs à garder. Un bon attribut divise les exemples en ensembles homogènes c.à.d qu'ils ne contiennent que des observations appartenant à la même classe, alors qu'un attribut inutile laissera les exemples avec presque la même proportion de valeur pour la variable cible .

Ce dont on a besoin c'est d'une mesure formelle de "bon" et "inutile". Pour cela, il existe des métriques standards homogénéisées avec lesquels on peut mesurer l'homogénéité d'un ensemble. Les plus connus sont l'indice de diversité de Gini et l'entropie [19].

En général l'entropie d'une variable aléatoire V avec des valeurs v_k chacune avec une probabilité $P(v_k)$ est définie comme :

$$\text{Entropie : } H(V) = - \sum_K P(v_k) \log_2 P(v_k) \quad (1.1)$$

— Avantages :

- C'est un modèle boîte blanche, simple à comprendre et à interpréter.
- Peu de préparation des données.
- Les variables prédictives en entrée peuvent être aussi bien qualitatives que quantitatives.
- Performant sur de grands jeux de données

Example	Input Attributes										Goal
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
x_1	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	$y_1 = \text{Yes}$
x_2	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	$y_2 = \text{No}$
x_3	No	Yes	No	No	Some	\$	No	No	Burger	0-10	$y_3 = \text{Yes}$
x_4	Yes	No	Yes	Yes	Full	\$	Yes	No	Thai	10-30	$y_4 = \text{Yes}$
x_5	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	$y_5 = \text{No}$
x_6	No	Yes	No	Yes	Some	\$\$	Yes	Yes	Italian	0-10	$y_6 = \text{Yes}$
x_7	No	Yes	No	No	None	\$	Yes	No	Burger	0-10	$y_7 = \text{No}$
x_8	No	No	No	Yes	Some	\$\$	Yes	Yes	Thai	0-10	$y_8 = \text{Yes}$
x_9	No	Yes	Yes	No	Full	\$	Yes	No	Burger	>60	$y_9 = \text{No}$
x_{10}	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10-30	$y_{10} = \text{No}$
x_{11}	No	No	No	No	None	\$	No	No	Thai	0-10	$y_{11} = \text{No}$
x_{12}	Yes	Yes	Yes	Yes	Full	\$	No	No	Burger	30-60	$y_{12} = \text{Yes}$

FIGURE 1.14 – L'ensemble d'apprentissage contient 12 observations décrites par 10 variables prédictives et une variable cible. [1]

— Inconvénients

- L'existence d'un risque de sur-apprentissage si l'arbre devient très complexe.
- On utilise des procédures d'élagage pour contourner ce problème.

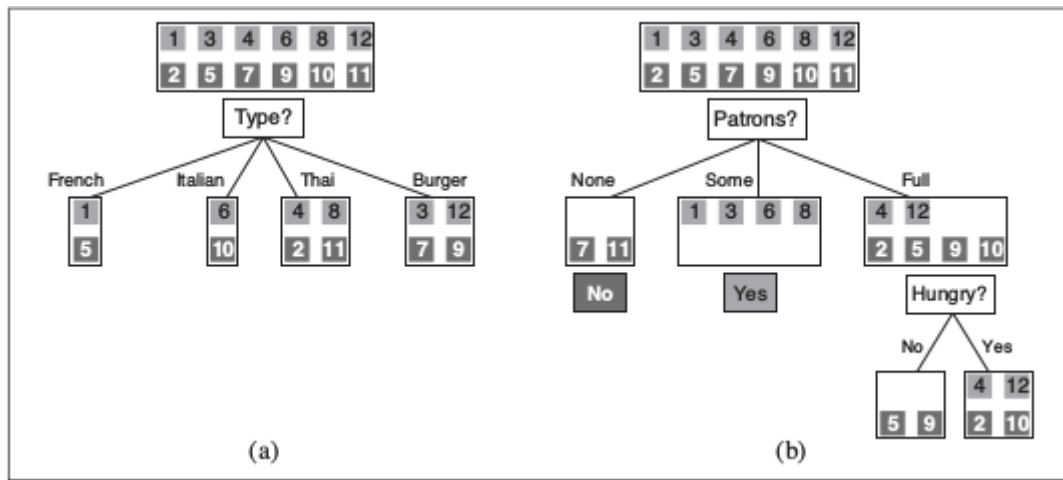


FIGURE 1.15 – Les exemples positifs sont représentés par des cases claires alors que les exemples négatifs sont représentés par des cases sombres. (a) montre que la division par l'attribut Type n'aide pas à avoir une distinction entre les positifs et les négatifs exemples. (b) montre qu'avec la division par l'attribut Patrons on obtient une bonne séparation entre les deux classes. Après division par Patrons, Hungry est un bon second choix. [1]

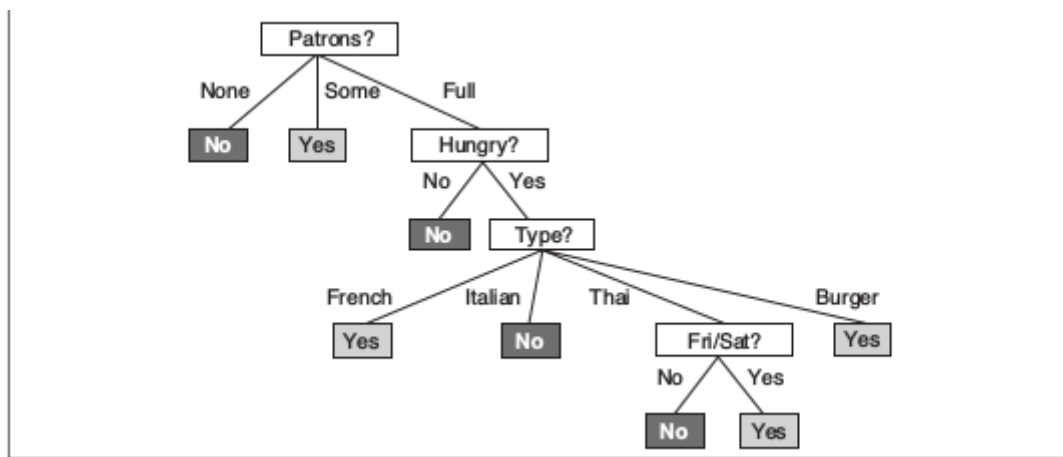


FIGURE 1.16 – L'arbre de décision déduit à partir des 12 exemples d'apprentissage.[1]

1.4.4 L'Algorithme AdaBoost

Le boosting est une technique d'ensemble qui tente de créer un classificateur fort à partir d'un certain nombre de classificateurs faibles. Cela se fait en construisant un modèle à partir des données d'apprentissage, puis en créant un deuxième modèle qui tente de corriger les erreurs du premier modèle. Les modèles sont ajoutés jusqu'à ce que l'ensemble d'apprentissage soit parfaitement prévu ou jusqu'à ce qu'un nombre maximal de modèles soit ajouté.

AdaBoost a été le premier algorithme de boosting réellement réussi développé pour la classification binaire. C'est le meilleur point de départ pour comprendre stimuler. Les méthodes de boosting modernes reposent sur AdaBoost, notamment les machines à boosting de gradient stochastique.

Algorithm Adaboost - Example

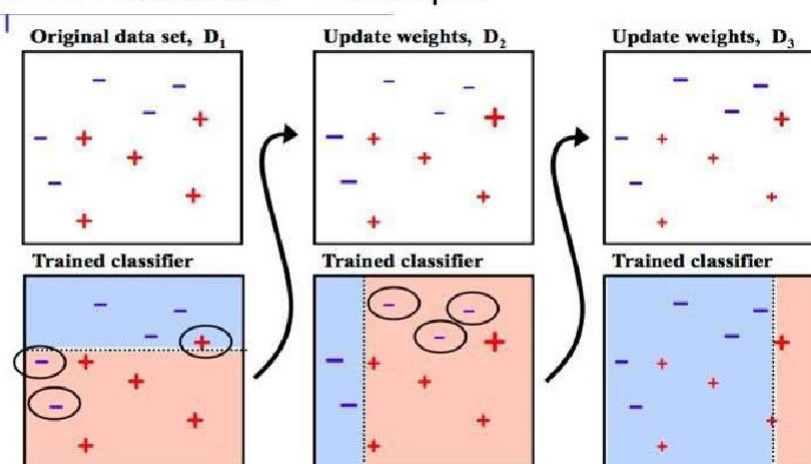


FIGURE 1.17 – Algorithme AdaBoost

AdaBoost est utilisé avec des arbres de décision courts. Une fois le premier arbre créé, les performances de l'arbre sur chaque instance d'entraînement servent à pondérer le degré d'attention accordé à l'arbre suivant créé qui doit prêter attention à chaque instance d'entraînement. Les données d'entraînement qui sont difficiles à prédire ont plus de poids, alors que les cas faciles à prévoir ont moins de poids. Les modèles sont créés séquentiellement l'un après l'autre, chacun mettant à jour les pondérations sur les instances d'apprentissage qui affectent l'apprentissage effectué par l'arborescence suivante de la séquence. Une fois toutes les arborescences construites, des prédictions sont établies pour les nouvelles données et la performance de chaque arborescence est pondérée par son degré de précision par rapport aux données d'apprentissage.

étant donné que l'algorithme accorde beaucoup d'attention à la correction des erreurs, il est important que vous disposiez de données claires, sans données aberrantes.

1.4.5 Méthode fondée sur l'arbre, J48

L'apprentissage par l'arbre est basé sur des arbres de décision issus d'un ensemble de formations qui sont labellisées. La prédiction de la décision ou de l'étiquette de classe (c'est-à-dire en termes d'apprentissage machine) peut être prise de la racine à un nœud de feuille. Elle peut traiter des données multidimensionnelles ainsi que d'apprendre à partir de ces données [52]. Cet apprentissage est basé sur des arbres de décision issus des données de formation de la classe. Il ressemble à un organigramme comme une structure arborescente. Dans la structure arborescente, les nœuds internes indiquent le test des attributs. Les branches sont les résultats des tests et les feuilles sont les étiquettes de classe. La racine représente le nœud le plus élevé de l'arbre. Dans la littérature, il existe de nombreux algorithmes d'arbre de décision énumérés ici.

1. ID3 (c'est-à-dire le dichotomiseur itératif 3 (ID3), c'est-à-dire qui se trouve dans [76])
2. C4.5
3. CART (c'est-à-dire arbre de classification et de régression également appelé analyse discriminante optimale hiérarchique)
4. CHAID (c'est-à-dire un détecteur automatique d'interaction CHi-carré dont les sorties sont très visuelles)
5. MARS (c'est-à-dire Multivariate Adaptive Regression Splines, qui est une technique de régression non paramétrique)

La méthode arborescente, J48 (c'est-à-dire la mise en œuvre de C4.5) est largement utilisée dans différentes applications de domaine (par exemple, le diagnostic d'un problème, l'astronomie, l'intelligence artificielle, l'analyse financière bancaire, la biologie moléculaire, et autres). Un arbre de décision est également utilisé pour produire un classificateur applicable et un apprentissage pour la prédiction du problème assigné [77] ; ce sont des sujets de recherche importants ces derniers jours. En même temps, il montre des performances significatives avec des analyses à variables multiples. Il est capable de sélectionner des caractéristiques complexes avec des caractéristiques liées à des règles [77]. En plus de divers points de données, il fonctionne avec des variables de données bruyantes et incomplètes (c'est-à-dire des valeurs manquantes). Il est simple et rapide d'apprentissage et de classification à partir des données de formation avec une bonne précision [78].

Malgré ces facilités, l'information qui sort n'est pas exacte et peut être influencée par d'autres domaines. Dans la plupart des cas, il faut une variable cible pour faire une prédiction dans les exemples de formation [52]. Elle est trop sensible sur des données irrévérencieuses et bruyantes qui contiennent dans la plupart des cas pratiques. Avec l'exploitation des exemples d'entraînement, le bruit ou les valeurs aberrantes peuvent se reproduire lors de la construction d'arbres, ce qui peut constituer une autre lacune des arbres de décision [78].

1.4.6 Random Forest

Random Forest est l'un des algorithmes d'apprentissage automatique les plus populaires et les plus puissants. Il s'agit d'un type d'algorithme d'apprentissage automatique appelé «Bootstrap Aggregation» ou «bagging».

Le bootstrap est une méthode statistique puissante permettant d'estimer une quantité à partir

1.4. LES DIFFÉRENTS TYPE D'ALGORITHME DE MACHINE LEARNING

d'un échantillon de données. Comme un moyen. Vous prenez beaucoup d'échantillons de vos données, calculez la moyenne, puis faites la moyenne de toutes vos valeurs moyennes pour vous donner une meilleure estimation de la vraie valeur moyenne.

Dans la mise en sac, la même approche est utilisée, mais plutôt pour estimer des modèles statistiques entiers, le plus souvent des arbres de décision. Plusieurs échantillons de vos données d'entraînement sont prélevés, puis des modèles sont construits pour chaque échantillon de données. Lorsque vous devez effectuer une prévision pour les nouvelles données, chaque modèle en fait une prédiction et la moyenne des prédictions est calculée afin de fournir une meilleure estimation de la valeur de sortie réelle.

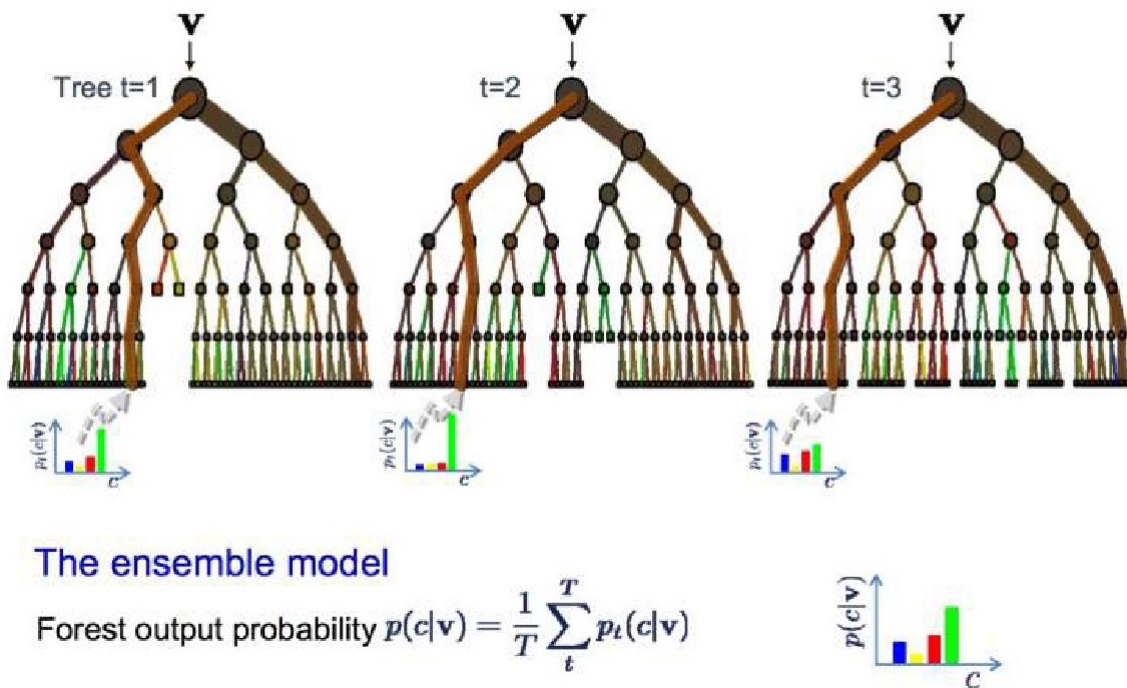


FIGURE 1.18 – Algorithme de Random Forest

Random forest est un "tweak" sur cette approche où les arbres de décision sont créés de telle sorte que plutôt que de sélectionner les points de partage optimaux, les séparations sous-optimales sont effectuées en introduisant aléatoire.

Les modèles créés pour chaque échantillon de données sont donc plus différents qu'ils ne le seraient autrement, mais restent précis de manière unique et différente. En combinant leurs prévisions, on obtient une meilleure estimation de la véritable valeur de sortie sous-jacente. Si vous obtenez de bons résultats avec un algorithme à variance élevée (comme les arbres de décision), vous pouvez souvent obtenir de meilleurs résultats en ensachant cet algorithme.

— Avantages :

- Très précis
- bon point de départ pour résoudre un Problème
- Flexible et peut s'adapter à une variété de données différentes
- Rapide à exécuter

- Facile à utiliser
- Utile pour les problèmes de régression et de classification
- Peut modéliser les valeurs manquantes
- Haute performance

- Inconvénients :
 - Lent à l'entraînement
 - Sur-adapter
 - Ne convient pas aux petits échantillons
 - Petit changement dans les données d'entraînement => changement de modèle
 - Parfois trop simple pour des problèmes très complexes

Conclusion

Le Machine Learning est un sujet vaste en évolution permanente. Les algorithmes qu'il met en ?uvre ont des sources d'inspiration variées qui vont de la théorie des probabilités aux intuitions géométriques en passant par des approches heuristiques.

1.5 Prédiction de la qualité du logiciel , travaux menés et méthodes employées

Les données de mesure du logiciel sont généralement utilisées pour la compréhension du logiciel. Mais dans plusieurs situations, on a besoin de prédire et d'évaluer la qualité d'un logiciel non encore en fonctionnement. Ceci est d'autant plus crucial quand le logiciel concerne un domaine d'application critique tel que l'industrie aéronautique. Pour ce domaine de l'industrie on a besoin de logiciels de haute fiabilité. En général, le développement de ce type de logiciels est tributaire du temps et du coût. Néanmoins, il faut s'assurer de la fiabilité désirée le plus tôt possible et certainement avant la mise en production. Paradoxalement, la fiabilité d'un logiciel est définie à partir de sa performance opérationnelle qui ne peut être mesurée qu'après une période de fonctionnement du logiciel. Ainsi, un recours à la prédiction s'impose pour évaluer la fiabilité du logiciel en cours de développement. Dans ce cas, la fiabilité est prédite à partir des indicateurs qui représentent notre compréhension du logiciel et qui sont disponibles dès les premières étapes du développement. De même on peut utiliser la prédiction lors de l'activité de maintenance et d'évolution du logiciel.

La prédiction trouve d'autres applications dans d'autres aspects de la qualité, tels que le coût, l'effort et la durée de développement d'un logiciel. Dans la littérature, différentes motivations justifient les efforts investis dans la prédiction.[20] et Khoshgoftaar [?] croient que la prédiction permet de ménager l'effort et d'épargner temps et argent car la correction retardée des fautes (après déploiement du logiciel) est une tâche très souvent coûteuse. Par conséquent, l'identification des modules susceptibles d'avoir un nombre élevé de défauts durant le développement permet d'économiser l'effort du développement et de la maintenance, tout en se concentrant sur le perfectionnement de la qualité de ces modules à haut risque. L'importance de la prédiction a attiré l'attention de plusieurs chercheurs qui ont essayé de proposer un

formalisme définissant d'une manière rigoureuse les concepts de la prédiction.

On trouve dans la littérature plusieurs travaux qui présentent des résultats et des expériences de prédiction de variables, associées à un facteur donné de la qualité du logiciel. Les facteurs qui ont attiré l'intérêt de nombreux groupes de recherches sont l'effort et la fiabilité.

Le gestionnaire et le chef du projet ont de nombreuses motivations pour prédire le coût d'un projet ainsi que son effort et sa durée, dès les premières étapes du cycle de vie d'un logiciel. Ils veulent planifier leurs tâches et allouer des ressources dans le but de diriger la gestion vers l'optimisation du coût. Par conséquent, plusieurs modèles d'estimation du coût et de l'effort ont été proposés comme COCOMO de Boehm [21, 22].

La construction de modèles prédictifs de la qualité du logiciel est une tâche complexe qui peut impliquer plusieurs techniques dans le but de produire des modèles ayant une bonne prédiction. Ces techniques peuvent être basées sur l'opinion d'experts, sur l'analogie, sur la décomposition, sur les statistiques ou sur des méthodes d'apprentissage. Nous nous intéressons, dans ce présent travail, aux deux dernières familles de techniques, vu leur utilisation répandue. En effet, les techniques de construction sont empruntées généralement au domaine de la statistique ou au domaine de l'apprentissage. Les techniques statistiques essaient, dans l'ensemble, de trouver une formule mathématique explicite qui exprime la relation entre les variables d'entrée d'un modèle et ses variables de sortie. Les modèles qui en résultent sont basés généralement sur des équations de régression. Ces techniques sont très utilisées dans la prédiction en général et dans celle visant des facteurs économiques et financiers en particulier. Ce type de prédiction aide à comprendre le marché et par conséquent à prendre des décisions de type A quel moment dois-je me retirer du marché. Cette tendance a influencé la prédiction de la qualité du logiciel. En effet, les premiers travaux de prédiction ont concerné les facteurs économiques du logiciel telle que l'estimation du coût. Parmi ce type de techniques, on trouve la régression linéaire, avec ses variétés.

1.5.1 Quelques travaux importants dans le domaine de la prédiction

1.5.2 QUANTIFICATION DES PARAMÈTRES DE QUALITÉ

Actuellement, la quantification des paramètres affectant la qualité des logiciels est un domaine important de la recherche en génie logiciel (SWE). Différents paramètres sont quantifiés dans diverses études en raison de leurs effets sur la qualité des logiciels. Les modèles de qualité des logiciels peuvent être classés en deux catégories : les modèles fixes et les modèles "définissez votre propre modèle" [8].

L'approche du modèle fixe

Ce type de modèles spécifie un ensemble particulier des caractéristiques de qualités où l'ensemble des attributs identifiés par le client est un sous-ensemble dans le modèle prédéfini. Afin de mesurer et de contrôler un attribut de qualité particulier, il est nécessaire d'utiliser les sous-caractéristiques, les mesures et les relations associées avec le modèle fixe.

Le modèle de Boehm [4] a été défini pour fournir un ensemble de "caractéristiques bien définies et bien différenciées de la qualité des logiciels". Ce modèle est considéré comme un

modèle hiérarchique, dans lequel les différents critères de qualité du modèle sont subdivisés au fur et à mesure que la hiérarchie du modèle s'étend. Deux niveaux de critères de qualité sont identifiés, de sorte que le niveau intermédiaire est encore divisé en caractéristiques mesurables. Le modèle McCall [5] était destiné aux développeurs de systèmes et devait être utilisé pendant le processus de développement. Le modèle identifie trois domaines de travail du logiciel : le fonctionnement du produit, la révision du produit et la transition du produit. Il reflète les priorités des développeurs ainsi que les points de vue des utilisateurs. Dromey [7] a proposé un modèle pour la qualité des produits logiciels. Ce modèle établit le lien entre les caractéristiques du produit et des attributs de qualité moins tangibles. En d'autres termes, le modèle illustré fournit un processus explicite pour intégrer des propriétés porteuses de qualité dans les logiciels. à leur tour, ces propriétés impliquent des attributs de qualité spécifiques. En outre, le modèle peut aider à mener une recherche systématique des défauts de qualité dans les logiciels.

Lamouchi et ses collaborateurs [9] ont quantifié les facteurs de qualité des logiciels par un modèle hiérarchique. Les facteurs sont subdivisés en critères et sous-critères à différents niveaux. Le dernier niveau a quantifié différentes mesures logicielles affectant les différents facteurs. Srivastava et ses collaborateurs [10] ont quantifié les paramètres de la qualité du logiciel selon différentes perspectives : celle du développeur, du gestionnaire de projet et de l'utilisateur. Ils ont pris en compte la moyenne pondérée des différents facteurs pour obtenir la qualité réelle du logiciel. Kanellopoulos et al (11) ont évalué la qualité du code source et le comportement statique d'un système logiciel, sur la base de la norme ISO/IEC-9126 (6), à l'aide du modèle AHP (Analytical Hierarchy Process)

L'approche "Définissez votre propre modèle"

Cette approche contraste avec l'approche des modèles fixes en ce sens qu'aucun attribut de qualité spécifique n'est défini. En revanche, un consensus sur les attributs de qualité pertinents est reconnu pour un système spécifique en coopération avec l'utilisateur. Ensuite, les attributs définis sont décomposés en caractéristiques de qualité qui peuvent être mesurées et leurs paramètres. La décomposition peut être guidée par un modèle de qualité existant. Les relations entre les attributs et les caractéristiques de qualité pourraient alors être définies de deux manières. Premièrement, par un modèle directement défini par les acteurs du projet. Deuxièmement, par un modèle défini indirectement qui peut être généré automatiquement. Les modèles directement définis sont considérés comme des graphiques de dépendance. Des exemples de ces approches ont été présentés dans [12, 13]. Samoladas et al. [12] ont présenté un modèle de qualité hiérarchique pour évaluer le code source ainsi que les processus communautaires. Lazić et al. [13] ont proposé un modèle qui peut être utilisé pour suivre les décisions de conception ainsi que les alternatives potentielles. Cela peut aider à réduire les coûts lors de l'évaluation des différentes alternatives de conception et lors du passage d'une alternative à l'autre.

Les modèles définis indirectement - proviennent à l'origine de deux domaines principaux, à savoir l'intelligence artificielle et les mathématiques. Le modèle de qualité pourrait être influencé par la technique choisie et son paramètres. Néanmoins, la technique utilisée n'a pas d'impact direct sur le modèle de qualité des résultats. En effet, les relations de qualité

représentées dans ces modèles sont compliquées, ce qui affecte négativement la compréhension des parties prenantes du projet.

1.5.3 PRÉVISION DE LA QUALITÉ DES LOGICIELS BASÉE SUR L'APPRENTISSAGE MACHINE

Les techniques de ML ont été utilisées dans de nombreux domaines de problèmes différents car ce domaine se concentre sur la construction d'algorithmes qui ont la capacité d'améliorer leurs performances automatiquement par l'expérience. L'application des techniques de ML à l'ingénierie logicielle a donné des résultats prometteurs et encourageants [14].

La machine à vecteur de soutien (SVM) a été appliquée avec succès dans de nombreux modèles de prévision de l'ingénierie logicielle. Le SVM a été utilisé pour la prédiction de l'effort de maintenance [15]. De plus, différentes études ont utilisé le SVM pour la prédiction des modules de prédictivité des défauts du logiciel [16-18].

Le réseau bayésien (BN) a été utilisé dans diverses études dans le domaine de l'ingénierie logicielle en raison de sa capacité à intégrer à la fois des données empiriques et des avis d'experts. Certaines études ont porté sur la prévision de la qualité des logiciels [19] et sur les efforts de développement [20]. Wagner [21] a proposé un cadre pour la création d'un BN pour la prévision de la qualité des logiciels en utilisant les modèles de qualité basés sur les activités. De même, Radliński [22] a proposé un modèle BN pour la prédiction intégrée de la qualité des logiciels. L'auteur a ensuite amélioré le cadre proposé en utilisant à nouveau le BN [23]. D'autres études ont utilisé le BN pour évaluer et prédire la maintenabilité du logiciel [24].

Kanmani et al (25) ont introduit l'utilisation des réseaux neuronaux (NN) comme outil de prédiction des défauts des logiciels. Yang et al. [26] a également proposé un modèle de prévision de la qualité des logiciels basé sur un NN flou pour identifier les erreurs de conception des produits logiciels dans les premières étapes du cycle de vie d'un logiciel. De plus, les NN ont été utilisés pour prédire à un stade précoce des attributs de qualité spécifiques, par exemple la fiabilité [27] et l'effort de maintenance [28].

Comme la relation entre les attributs de qualité internes et externes est entourée d'impression et d'incertitude, différentes études dans la littérature ont tenté d'utiliser la capacité de la logique floue (FL) pour estimer la qualité du logiciel. Mittal et ses collaborateurs [29] ont proposé une approche basée sur la FL pour quantifier la qualité des logiciels, où les logiciels examinés ont reçu des notes de qualité sur la base de deux mesures. Srivastava et ses collaborateurs (30) ont essayé de classer la qualité des logiciels en utilisant l'approche floue à critères multiples. Cette approche a permis de classer les logiciels sur la base des documents de spécifications des exigences logicielles. Ils ont réalisé une analyse similaire en utilisant le modèle de qualité ISO/IEC 9126 [31]. Yang [32] a proposé une approche pour mesurer la qualité des produits logiciels avec les normes ISO basées sur la technique FL. Un modèle AHP flou a été proposé par Yuen et al (33) pour évaluer la qualité du logiciel et pour sélectionner le fournisseur du logiciel en cas d'incertitude. Le modèle classe les différents logiciels pour permettre de sélectionner le meilleur de manière appropriée. Dans un travail supplémentaire [34], ils ont utilisé le modèle AHP flou et plus particulièrement la méthode des moindres carrés logarithmiques flous pour estimer la qualité des logiciels. Mago et al. [35] a utilisé FL pour analyser la qualité de la conception de logiciels orientés objet.

Pierre Oum Sack propose un modèle d'évaluation de la qualité basé sur l'Approche à base

d'apprentissage automatique et de transformation de modèles. Ce modèle permet de prédire la maintenabilité des logiciels à base des métriques logiciels.

Salma Hamza propose dans sa thèse Une approche pragmatique pour mesurer la qualité des applications à base de composants logiciels. ce modèle permet de prédire la qualité des logiciels à partir des métriques dans le domaine composant tel que les métriques au niveau composants , les métriques au niveau application et les métriques au niveau interface.

Diverses études ont été consacrées à la prédiction de la maintenabilité des logiciels. De nombreuses approches basées sur le FL ont été proposées pour mesurer les attributs de qualité après coup. Par exemple, Mittal et al (36) ont proposé une approche basée sur le FL pour évaluer la productivité de la maintenance des systèmes logiciels. Sharam et al (37) ont introduit une approche basée sur le FL pour la prédiction de la maintenabilité des systèmes basés sur des composants. De plus, une autre approche basée sur le FL a été proposée par Singh et al (38) pour prédire la maintenance des logiciels. Dans notre travail précédent [39], nous avons présenté les premières expériences de construction de modèles basés sur le FL pour prédire la maintenabilité des logiciels. D'autres chercheurs ont envisagé différentes techniques de FL pour prédire la maintenabilité du logiciel. De même, différentes approches de FL ont été introduites pour évaluer différents attributs de qualité tels que la facilité d'utilisation [40], la compréhensibilité [41] et la réutilisabilité [42, 43], et la fiabilité [44].

Il convient de noter ici qu'aucune des tentatives évoquées ci-dessus n'a considéré le modèle de transparence comme un objectif. à notre connaissance, les modèles proposés précédemment n'étaient pas suffisamment transparents pour que l'homme puisse y intégrer ses connaissances.

Conclusion

Dans cette partie nous avons parlé de l'apprentissage automatiques et présenté quelques algorithmes. Nous avons montré son utilisation dans la prédiction de la qualité du logiciel, par l'étude de différents travaux effectués dans le domaine. Nous avons après une critique de l'état l'art.

PRÉ-TRAITEMENT DES DONNÉES ET CONSTRUCTION DU JEU DE DONNÉES

SOMMAIRE

2.1	Introduction	45
2.2	Caractéristiques à évaluer	45
2.3	Extraction du programme	53

2.1 Introduction

Dans ce chapitre, nous avons discuté de la façon dont les ensembles de données sont préparés en utilisant les mesures Chidamber et Kemerer.

2.2 Caractéristiques à évaluer

Ici il est question de bien clairement définir les caractéristiques ou attributs de qualité que nous souhaitons évaluer. Selon notre modèle de qualité choisi notamment le modèle ISO 9126, la qualité du logiciel est subdivisé en 6 principales caractéristiques ou attributs à savoir :

Attribut	Définition
Capacité fonctionnelle	La capacité du produit logiciel à fournir des fonctions qui répondent aux besoins déclarés et implicites lorsque le logiciel est utilisé dans des conditions précises.
Fiabilité	La capacité du produit logiciel à maintenir un niveau de performance spécifié lorsqu'il est utilisé sous conditions spécifiées.
Facilité d'utilisation	la capacité du produit logiciel à être compris, appris et attrayant pour les utilisateurs, selon des circonstances spécifiques d'utilisation
Efficacité	la capacité du produit logiciel à fournir le niveau attendu de performances, en fonction des ressources utilisées selon des conditions fixées

2.2. CARACTÉRISTIQUES À ÉVALUER

Maintenabilité	la capacité du produit logiciel à être modifiée. Les modifications incluent les corrections, les améliorations et l'adaptation lors d'un changement d'environnement, d'exigences ou de spécifications fonctionnelles
Portabilité	la capacité d'une application à basculer d'un environnement à un autre

TABLE 2.1 – Attributs de qualité du modèle ISO 9126

Chacune de ces attributs de qualité est subdivisée en sous-caractéristiques suivant le tableau :

Les caractéristiques	Sous-caractéristiques	Définition
Capacité fonctionnelle	Pertinence	La capacité du produit logiciel à fournir un ensemble approprié de fonctions pour des tâches spécifiques et les objectifs des utilisateurs.
	Précision	La capacité du produit logiciel à fournir les résultats ou effets corrects ou convenus avec les degrés de précision.
	L'interopérabilité	La capacité du produit logiciel à interagir avec un ou plusieurs systèmes spécifiques
	Conformité	La capacité du produit logiciel à respecter les normes, conventions ou règlements dans les lois et des prescriptions similaires relatives à la fonctionnalité.
	Sécurité	La capacité du produit logiciel à protéger les informations et les données de sorte que les personnes non autorisées ou Les systèmes ne peuvent pas les lire ou les modifier et les personnes ou systèmes autorisés ne se voient pas refuser l'accès à ces systèmes.
Fiabilité	Maturité	La capacité du produit logiciel à éviter les défaillances dues à des défauts du logiciel.
	Tolérance aux pannes	La capacité du produit logiciel à maintenir un niveau de performance déterminé dans le cas d'un logiciel ou de violation de son interface spécifiée.
	Possibilité de récupération	La capacité du produit logiciel à rétablir un niveau de performance spécifié et à récupérer les données directement affectées en cas de panne.

2.2. CARACTÉRISTIQUES À ÉVALUER

Les caractéristiques	Sous-caractéristiques	Définition
Facilité d'utilisation	Compréhensibilité	La capacité du produit logiciel à permettre à l'utilisateur de comprendre si le logiciel est adapté, et comment il peut être utilisé pour des tâches et des conditions d'utilisation particulières.
	Facilité d'apprentissage	La capacité du produit logiciel à permettre à l'utilisateur (novice, expert) d'apprendre son application
	Opérabilité	La capacité du produit logiciel à permettre à l'utilisateur de l'exploiter et de le contrôler.
Efficacité	efficacité en temps	La capacité du produit logiciel à fournir des temps de réponse et de traitement et des débits appropriés lors de l'exécution de sa fonction, dans des conditions déterminées.
	efficacité en ressources	La capacité du produit logiciel à utiliser des quantités et des types de ressources (à savoir l'utilisation de la mémoire, du processeur, du disque et du réseau.) appropriés lorsque le logiciel remplit sa fonction dans des conditions déterminées.
Maintenabilité	Analysabilité	La capacité du produit logiciel à être diagnostiqué pour les déficiences ou les causes de défaillances du logiciel, ou à identifier les parties à modifier.
	Changeabilité	La capacité du produit logiciel à permettre la mise en œuvre d'une modification spécifique.
	La stabilité	La capacité du produit logiciel à éviter les effets inattendus des modifications du logiciel.
	Testabilité	la capacité du produit logiciel à permettre la validation d'un logiciel modifié.
Portabilité	Adaptabilité	La capacité du produit logiciel à être adapté à différents environnements spécifiés sans appliquer des actions ou des moyens autres que ceux prévus à cet effet pour le logiciel considéré.
	Installabilité	La capacité du produit logiciel à être installé dans un environnement spécifique
	Conformité aux règles de portabilité	La capacité du produit logiciel à respecter les normes ou conventions relatives à la portabilité. Un exemple serait la conformité Open SQL qui concerne la portabilité de la base de données utilisée.
	interchangeabilité	La capacité du produit logiciel à être utilisé à la place d'un autre produit logiciel spécifié pour le même objectif dans le même environnement.

2.2. CARACTÉRISTIQUES À ÉVALUER

Les caractéristiques	Sous-caractéristiques	Définition
----------------------	-----------------------	------------

TABLE 2.2 – Le tableau complet des caractéristiques et sous-caractéristiques du modèle de qualité ISO 9126-1

La prochaine étape consiste à associer les métriques aux caractéristiques et sous-caractéristiques définies plus haut. La qualité est un concept générique associé à un produit logiciel. Les caractéristiques tels que la maintenabilité représentent les attributs de haut niveaux dans les modèles qualitatifs tel que la norme ISO 9126. Ces attributs ne sont pas directement mesurables et doivent être décomposé en sous-caractéristiques comme l'illustre le tableau précédent . Le processus de décomposition est appliqué jusqu'à l'obtention des entités attributs, qui elles sont mesurables. ces attributs peuvent être simple ou dérivé.

L'évaluation de la qualité dans ce mémoire s'appuie sur le modèle de qualité issue de la norme ISO 9126, ce modèle qui se concentre sur l'évaluation de la **qualité du produit** définit les facteurs assez précis pour atteindre cet objectif. Par la suite nous définirons de façon sommaire les sous-caractéristiques selon le modèle de qualité ISO 9126.

Sous-caractéristiques	Attributs	Propriétés de mesures
Modifiabilité	Structure du code	complexité, abstraction, encapsulation, taille du code
	Documentation	Commentaire
	Modularité	encapsulation, cohésion, couplage
	Localisation de changements	Cohésion, abstraction, polymorphisme
Testabilité	Structure	complexité, abstraction, héritage
	Concision	Complexité, cohésion, couplage
Stabilité	Encapsulation	Encapsulation
	Héritage	héritage
Analysabilité	Structure	Complexité
	Couplage	couplage

TABLE 2.3 – les caractéristiques et attributs de la maintenabilité

Sous-Caractéristiques	Attributs	Propriétés mesurées
	Documentation de qualité	Présence des documents de formation, système, etc.

2.2. CARACTÉRISTIQUES À ÉVALUER

Compréhensibilité	Documentation disponible	La qualité du manuel utilisateur (claire, complet, aide)
	Convention de codage	Nommage, structuration du code, etc.
Apprentissage	Temps d'utilisation	Niveau d'expertise (débutant, avancé, expert)
	Temps de configuration	Niveau d'expertise (débutant, avancé, expert)
	Temps d'expertise	Niveau d'expertise (débutant, avancé, expert)
	Temps d'administration	Niveau d'expertise (débutant, avancé, expert)
Opérabilité	Niveau de complexité	(débutant, avancé, expert)
	Interfaces fournies	Présence
	Interfaces requises	Présence
	Effort d'utilisation	Niveau d'expertise (débutant, avancé, expert)

TABLE 2.4 – Les caractéristiques et attributs de l'utilisabilité

Sous-Caractéristiques	Attributs	Propriétés mesurées
Maturité	Volatilité	Mesure le temps moyen entre les différentes versions du logiciel
	Evolutibilité	C'est la mesure des versions produites du logiciel
	Bugs traités	Mesure le nombre moyen des bogues traités dans les précédentes versions
Tolérance aux pannes*		
Capacité de récupération*	Sérialisation	Présence des mécanismes permettant la sérialisation
	Persistance	Présence de mécanismes de gestion de la persistance
	Transaction	Présence de mécanismes permettant la gestion des transactions
	Gestion des erreurs	Présence de mécanismes permettant la gestion des exceptions et des erreurs

TABLE 2.5 – Les caractéristiques de la fiabilité

Le logiciel étant une entité abstraite, l'élément qui reste incontournable pour son évaluation est son code source. D'après le tableau précédant nous avons raffiné les caractéristiques en sous-caractéristique puis chaque sous-caractéristique en entité mesurable au travers des métriques de code, ce qui fait passer la métrique à une position centrale dans le processus d'évaluation de la qualité du logiciel. Nous distinguerons principalement deux types de métriques : les métriques simples et les métriques dérivées. Une métrique sera dite simple lorsqu'elle indépendamment fonctionnelle des autres métriques, un exemple de métrique simple est la métrique SLOC qui compte le nombre de ligne de code pour une méthode, une classe ou encore même un paquetage. Une métrique dérivée se définit comme une fonction ou formule qui combine deux ou plusieurs métriques. Dans le modèle de qualité instancié par Bansiya dans Bansiya réussit à attribuer les mesures de conception aux propriétés de conception. Il combine les métriques de manière significative. Ainsi la validité des propriétés de conception dépend directement des mesures et de leur combinaison. Selon Bansiya une combinaison de métriques de code source doit être effectuée avec soin car plus les métriques sont combinées, moins l'influence d'une seule source peut être vérifiée de manière fiable. Il faut donc faire un compromis entre l'expressivité et la traçabilité d'une propriété de conception. Bansiya n'utilise qu'une seule mesure pour une propriété de conception, on peut donc dire qu'une bonne traçabilité est assurée. Dans le tableau suivant on peut voir quelles mesures sont utilisées pour évaluer les propriétés de conception :

Propriété	Définition	Exemple de mesure
Taille	Mesure par exemple le nombre de classe dans un projet	Nombres de classes
Hierarchies	Les hiérarchies sont utilisées pour représenter différents aspects de la généralisation-spécification. Mesure par exemple le nombre de classes non-héritées qui possèdent des classes filles dans un logiciel	Nombre hiérarchies
Abstraction	Mesure de l'aspect généralisation-spécification.	Nombre d'ancêtres
Encapsulation	Elle définit le comportement d'un objet pendant sa création ou son utilisation. Dans le paradigme objet mesure la visibilité des déclarations (attributs et méthodes) par les autres objets	Mesure d'accès aux données
Couplage	Mesure de l'interdépendance d'un objet dans un autre objet de la conception. Mesure également le nombre d'objets qui sont accédés par un objet pour le fonctionnement de ces derniers	Couplage direct entre classe

2.2. CARACTÉRISTIQUES À ÉVALUER

Cohésion	Mesure les relations connexes entre les attributs et les méthodes dans une classe	Cohésion entre les méthodes dans une classe
Composition	Mesure les relations <i>Est une partie de, consiste en</i> , ce sont les relations d'agrégations dans le paradigme objet	Mesure de l'agrégation
Héritage	C'est la mesure des relations d'héritages (est un) entre les classes. Cette relation est liée au niveau d'imbrication des classes dans l'hiérarchie d'héritages	Mesure de l'abstraction fonctionnelle
Complexité	Mesure le degré de difficulté dans la compréhension et la structuration interne ou externe des classes et de leur relation	Nombre de méthodes
Polymorphisme	Mesure les services (méthodes) qui sont dynamiquement déterminés au moment de l'exécution	Nombre de méthodes polymorphiques
Appel et utilisation (messagerie)	Mesure le nombre des méthodes publiques qui sont visibles et disponibles par les autres classes, mesure également les services fournies	Taille des méthodes dans une interface

TABLE 2.6 – Tableau des propriétés conception mesurées

Bansiya utilise des propriétés de conception pondérées pour construire un attribut de qualité. Il met sur pied plusieurs formules donc les pondérations et la combinaison des propriétés sont résumés dans le tableau suivant.

Attribut de qualité	Calcul de l'indice
Formule 1 : Réutilisabilité	$= 0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 * DesignSize$
Formule 2 : Flexibilité	$= 0.25 * Encapsulation + 0.25 * Coupling + 0.5 * Composition + 0.5 * Polymorphism$
Formule 3 : Compréhensibilité	$= 0.33 * Abstraction + 0.33 * Encapsulation + 0.33 * Coupling + 0.33 * Cohesion + 0.33 * Polymorphism + 0.33 * Complexity + 0.33 * DesignSize$
Formule 4 : Fonctionnalité	$= 0.12 * Cohesion + 0.22 * Polymorphism + 0.22 * Messaging + 0.22 * DesignSize + 0.22 * Hierarchies$
Formule 5 : Extensibilité	$= 0.5 * Abstraction + 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism$

2.2. CARACTÉRISTIQUES À ÉVALUER

Formule 6 : Effectivité	= $0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism$
-------------------------	--

TABLE 2.7 – Formules de calcul des attributs de qualité de Bansiya

Comme on peut le voir, les pondérations peuvent être positives ou négatives. Le signe algébrique indique que la propriété de conception spécifiée a une influence respectivement positive ou négative sur l'attribut de qualité. Par exemple, la réutilisabilité est positivement influencée par la taille du design (plus il y a de classes, plus on peut réutiliser), la cohésion (plus le design est cohésif, plus il y a de modules qui peuvent être utilisés dans d'autres projets) et la messagerie (plus le design offre de services, plus il a de chances d'être utilisé dans un autre contexte). D'autre part, le couplage réduit la réutilisabilité (plus un objet est couplé à un autre, moins il est possible de l'utiliser dans un contexte différent). La somme des propriétés de conception pondérées se situe dans la plage de $[-1... + 1]$, de sorte que tous les attributs de qualité ont la même plage. Pour les influences positives, une valeur pondérée initiale de +1 ou +0,5 a été fixée. Pour les influences négatives, une valeur de -1 ou -0,5 a été choisie. Cette valeur a ensuite été modifiée proportionnellement de manière à ce que la somme des pondérations obtenues donne ± 1 .

Ainsi nous pouvons en déduire la formule suivante pour la maintenabilité :

Maintenabilité = $\alpha * Analysability + \beta * Changeability + \lambda * Stability + \Theta * Testability$ Où les poids $\alpha, \beta, \lambda, \Theta$ sont obtenus grâce au processus de pondération utilisé par Bansiya. Les métriques simples et dérivées sont décrites par une échelle et une unité. Une échelle est un ensemble ordonné de valeurs, continue ou discrète associée à une mesure. Une unité est normalement associée à une échelle. Quatre types d'échelles sont communément définies :

? Nominale : les valeurs mesurées sont catégorisées.

? Ordinale : les valeurs mesurées sont ordonnées, exemple la classification des pannes suivant leurs niveaux de sévérité.

? Intervalle : Les valeurs mesurées sont à égale distance si elles correspondent à des attributs de même distance. Par exemple la complexité cyclomatique a une valeur minimale, mais chaque incrémentation représente un chemin additionnel.

? Ratio : par exemple le ratio de bugs (pannes ou erreurs) détectés par un testeur de qualité est double : cela implique par exemple que le testeur est deux fois plus efficace.

Ces différentes sous-caractéristiques sont en fonction d'un ensemble de métriques. Les métriques sont choisies en fonction des attributs et propriétés mesurées décrits dans les travaux de thèse de Pierre Oum Sack et des articles "A mapping study on design-time quality attributes and metrics" et "Empirical Evidence on the Link between Object-Oriented Measures and External Quality Attributes" comme décrit dans le tableau. Dans ces articles les auteurs font un mapping entre les attributs de qualité de haut niveau tel que la maintenabilité avec les métriques de bas niveau en suivant une méthodologie de Goal Questions Metrics introduit par Basili. Cette méthodologie consiste dans un premier temps à formuler les Questions permettant de préciser les Attributs de qualités de haut niveau. Par la suite ils identifient plusieurs critères tel que les journaux ayant publié des articles liés à la qualité logicielle ou encore les articles comprenant les mots clés tel que "attribut de qualité" OU "caractéristique de qualité" OU "métrique de qualité" OU "métrique de logiciel" OU "mesure de logiciel" OU "exigence de qualité" OU "cadre de qualité"

2.3. EXTRACTION DU PROGRAMME

OU "exigence non fonctionnelle" OU "exigence non fonctionnelle", après la sélection des critères ils en ressortent avec plus de 2800 articles liés à la qualité logicielle, puis la dernière étape consiste à associer les attributs de qualités aux mesures suivant la figure ci-dessous :

Association between QAs and quality metrics.

Quality attributes	Quality metrics	Freq.	Quality attributes	Quality metrics	Freq.
Maintainability	Depth of Inheritance Tree (DIT) *	6	Change proneness	Depth of Inheritance Tree (DIT) *	3
	Lines of Code (LOC) *	6		Number of Children (NOCC) *	3
	Weighted Methods per Class (WMC) *	6		Coupling Between Objects (CBO) *	3
	Cyclomatic Complexity (CC-VG)	5		Response for Class (RFC) *	3
	Lack of Cohesion of Methods-1 (LCOM1) *	5		Lack of Cohesion of Methods-1 (LCOM1) *	3
	Tight Class Cohesion (TCC)	4		Data Abstraction Coupling (DAC) *	3
	Number of Children (NOCC) *	4		Number of Attributes (NA)	3
	Response for Class (RFC) *	4	Understand ability	Lines of Code (LOC) *	3
	Message Passing Coupling (MPC) *	4		Depth of Inheritance Tree (DIT) *	2
	Data Abstraction Coupling (DAC) *	4		External Class Complexity (ECC) *	2
Reusability	Number of Methods(NOM)	4		External Class Size (ECS) *	2
	Lack of Cohesion of Methods-1 (LCOM1)*	3	Testability	Response for Class (RFC) *	4
	Lines of Code (LOC) *	2		Coupling Between Objects (CBO) *	3
	Coupling Between Objects (CBO) *	2		Lack of Cohesion of Methods-1 (LCOM1) *	3
	Response for Class (RFC) *	2		Lines of Code (LOC) *	3
	Message Passing Coupling (MPC) *	2	Modifiability	Depth of Inheritance Tree (DIT) *	2
	Weighted Methods per Class (WMC) *	2		Halstead n1	2
	Number of Children (NOCC) *	2		Halstead n2	2
	External Class Complexity (ECC) *	2	Stability	Weighted Methods per Class (WMC) *	2
	External Class Size (ECS) *	2		Lines of Code (LOC) *	2
				System Design Stability (SDI)	2

* These terms are duplicate in the table.

FIGURE 2.1 – Mapping entre les attributs de qualité et les métriques du code

La caractéristique maintenabilité qui par exemple est $= \alpha * Analysability + \beta * Changeability + \lambda * Stability + \Theta * Testability$

ici $\alpha, \beta, \lambda, \Theta$ ont la valeur 1, on supposera pour cela que les sous-caractéristiques ont une influence équivalente. Ainsi dans ce travail nous utiliserons les formules suivantes pour les différentes sous-caractéristiques de la maintenabilité :

Analysabilité = $0.5 * WMC + 0.5 * CBO$

Modifiabilité = $0.25 * LCOM + 0.25 * DIT + 0.25 * LOC + 0.25 * CBO$

Stabilité = $0.5 * DIT + 0.5 * NOC$

Testabilité = $0.33 * DIT + 0.33 * CBO + 0.34 * LCOM$

Ici nos formules à discuter avec l'enseignant

2.3 Extraction du programme

Les données que nous avons utilisé dans un premier temps proviennent des ensembles PROMISE de la NASA. Nous avons utilisé l'ensemble de donnée provenant du dépôt KC1. La description détaillé de l'ensemble de donnée est contenu dans le tableau suivant. Cet ensemble de données est préparé avec l'extraction du code source en utilisant les métriques McCabe, Halstead, Chidamber et Kemerer. Pour comprendre les attributs de Halstead, nous considérons le programme C suivant :

```
main ()
```

```
{
```

```
float x, y, z, average ;

scanf( "%d %d %d", &x, &y, &z) ;

average = (x + y + z) / 3 ;

printf( "Average is %d", average) ;

}
```

Programme qui calcule la moyenne de 3 nombres

Pour l'extraction de ce code, nous utilisons les attributs HALSTEAD_OPERATOR, HALSTEAD_OPERANDS, HALSTEAD_PROGRAM_LENGTH, HALSTEAD_VOLUME. Dans ce programme le nombre n_1 d'opérateur est 10, nous avons **main, (,) , float, &, =, +, /, printf, et scanf**.

Les Opérandes uniques n_2 sont au nombre de 7, il s'agit de **x, y, z, average, %d, %d, %d, 3, ?Average is = %d?**.

Le nombre total d'opérateurs, N_1 est de 16 et le nombre total d'opérandes, N_2 est de 15, donc la taille du programme N est égale à 31.

Ainsi nous avons les mesures suivantes en ce qui concerne :

HALSTEAD_PROGRAM_LENGTH :

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2 = 10 \log_2 10 + 7 \log_2 7 = 52.9$$

$$\text{HALSTEAD_VOLUME, } V = N \log_2 n = 31 \log_2 17 = 126.7$$

2.3.1 Attributs et modules des ensembles de données

Le dépôt PROMISE de la NASA est un célèbre dépôt de données où des programmes ont été utilisés pour le système terrestre ou le système de satellites. Les programmes étaient écrits en C ou C++. Ces ensembles de données ont été créés avec l'extraction de programmes en utilisant des métriques. Dans cette section, nous décrivons les attributs de l'ensemble de données KC1 avec les métriques d'extraction. Dans ce travail nous utiliserons le dépôt KC1 Prévision des défauts des logiciels qui est un des ensembles de données de défaut du programme de données métriques(MDP) de la NASA. Cet ensemble de donnée est écrit en langage C++ et ses données provenant d'un logiciel de gestion du stockage pour la réception et le traitement des données au sol. Les données proviennent des extracteurs de code source de McCabe et Halstead. Ces caractéristiques ont été définies dans les années 70 pour tenter de caractériser objectivement les caractéristiques du code qui sont associées à la qualité des logiciels. Il contient 2109 modules ou instances et 22 attributs décomposé comme suit 5 mesures différentes de lignes de code , 3 mesures McCabe, 4 mesures Halstead de base, 8 mesures Halstead dérivées, un compte de branche et 1 champ de but)

Dans un second temps nous utilisons les données provenant de 3 projets logiciels tel que Apache ou encore Azeureus. Azeurus est un client torrent à l'instar de Utorrent ou Bit-Torrent. Azeurus est écrit en java et contient plus 4000 classes et plus de quatre cent mille lignes de code. Pour l'extraction du code source nous avons utilisés CK(Chidamber-Kemerer metrics). CK calcule des mesures de code au niveau de la classe et au niveau de la métrique

dans les projets Java au moyen d'une analyse statique (c'est-à-dire sans avoir besoin de code compilé). Actuellement, il contient un large ensemble de métriques, dont nous pouvons citer : **CBO (Couplage entre objets)** : Compte le nombre de dépendances d'une classe. Les outils vérifient tout type utilisé dans l'ensemble de la classe (déclaration de champ, méthode types de déclarations, déclarations variables, etc). Il ignore les dépendances à Java lui-même (par exemple `java.lang.String`).

DIT (Depth Inheritance Tree) : Il compte le nombre de "mères" que possède une classe. Toutes les classes ont un DIT d'au moins 1 (tout le monde hérite de `java.lang.Object`). Pour que cela se produise, des classes doivent exister dans le projet (c'est-à-dire si une classe dépend de X qui s'appuie sur un fichier jar/dépendance, et X dépend d'autres classes, DIT est compté comme 2).

Nombre de méthodes : compte le nombre de méthodes.

NOSI (Nombre d'appels statiques) : compte le nombre d'appels aux méthodes statiques.

RFC (Response for a Class) : compte le nombre d'appels de méthode uniques dans une classe. Comme les appels sont résolus via une analyse statique, cette implémentation échoue lorsqu'une méthode a des surcharges avec le même nombre de paramètres, mais des types différents.

WMC (Weight Method Class) ou la complexité de McCabe . Il compte le nombre d'instructions de branche dans une classe.

LOC (lignes de code) : il compte les lignes de code du programme, en ignorant les lignes vides et les commentaires (c'est-à-dire, c'est les lignes de code source ou SLOC).

LCOM (manque de cohésion des méthodes) : calcule la métrique LCOM.

TCC (Tight Class Cohesion) : mesure la cohésion d'une classe avec une plage de valeurs de 0 à 1.

LCC (Loose Class Cohesion) : Similaire au TCC mais il inclut en outre le nombre de connexions indirectes entre les classes visibles pour le calcul de la cohésion.

Dans cette chapitre chapitre nous avons identifiés les attributs de qualités, après les avoir décomposés en sous-caractéristiques ,nous les avons associés aux métriques de bas niveau afin qu'ils servent de variables d'entrée dans nos différents algorithmes d'apprentissage automatique



FIGURE 2.2 – Schema de décomposition des attributs de qualité

IMPLÉMENTATION : PRÉSENTATION DES DIFFÉRENTS OUTILS ET INTERPRÉTATION DES RÉSULTATS

SOMMAIRE

3.1	Introduction	56
3.2	Présentation du modèle de qualité	56
3.3	Présentation de l'extracteur des métriques CK(Chidamber-Kemerer metrics)	57
3.4	WEKA(Waikato Environment for Knowledge Analysis)	59

3.1 Introduction

3.2 Présentation du modèle de qualité

Dans ce travail nous souhaitons évaluer la qualité du produit logiciel , pour cela nous nous servons des métriques issues du code source. Notre modèle de qualité est calqué sur le modèle ISO 9126 qui se concentre sur la qualité du produit, la norme ISO 9126 ne repose pas uniquement sur les métriques et de nombreuses mesures doivent être manuelles.

La norme lie les qualités internes et externes d'un logiciel, indique un lien de causalité entre ces deux types de caractéristiques mais ne donne aucune indication quant aux bonnes ou mauvaises valeurs.

Cette norme semble être une bonne approche pour déterminer la qualité d'un logiciel dans son ensemble et fournir une vue globale satisfaisante. Cependant, la norme ne précise pas de manière explicite comment mesurer les caractéristiques qualité définies et comment les relier aux métriques de bas niveau.

Ainsi dans ce mémoire nous nous servons de la norme ISO 9126 comme socle pour l'évaluation de la qualité, c'est à partir de ce modèle que nous identifions les attributs de qualité de haut niveau. En suite notre modèle est inspiré de celui de Dromey qui représente essentiellement une configuration à trois niveaux avec des attributs de qualité, des propriétés de conception et des composants(classes , méthodes,paquetage). Son point de vue est axé sur la conception, et l'approche consistant à utiliser des mesures de produits est basée sur l'hypothèse que la mesure et le contrôle des propriétés internes des produits (indicateurs de qualité internes) se traduiront par un meilleur comportement externe des produits (absence

de défaillances, simplicité de changement, qualité d'utilisation) Enfin Bansiya dans ses travaux associe les métriques de qualité et les attributs de haut niveau ,son modèle se concentre sur une vue externe de la qualité, notre modèle de qualité est également inspiré de ce dernier, On peut donc dire que la norme ISO 9126 tente d'unir la vue interne et externe et la qualité telle que l'utilisateur la perçoit. Les modèles de Dromey et Bansiya prennent en compte la vue interne et externe. Cela signifie que seules les informations disponibles par le code source du produit lui-même sont prises en compte pour évaluer la qualité. De cette façon, une évaluation technique du logiciel en tant que produit est possible (syntaxe) mais ignorer le point de vue de l'utilisateur rend impossible l'évaluation de la qualité sémantique du produit. Avec cette réduction, l'utilisation de mesures du code source pour l'évaluation de la qualité devient applicable. Ainsi, l'évaluation de la qualité à l'aide du code source ne peut couvrir que la partie architecte/développeur de modèles de qualité complets comme la norme ISO 9126.

Ainsi dans notre modèle de qualité, nous commençons par sélectionner les attributs de qualité que nous souhaitons évaluer tel que la maintenabilité selon la norme ISO 9126, puis nous la raffinons en sous-caractéristiques toujours suivant la norme ISO 9126, en nous inspirant des modèles Dromey et Bansiya nous faisons une association entre les sous-caractéristiques de la qualité et les métriques de code. Ces métriques constitueront les entrées du système d'apprentissage pour la prédiction de la qualité du logiciel.

Dans notre travail nous évaluons la qualité d'un logiciel en nous servant de son code source, ce qui rend assez large notre champ d'action. Par la suite nous nous concentrons sur les logiciels écrits en langage de programmation orienté objet et plus particulièrement le langage Java. Java est reconnu pour sa portabilité sur toutes les plateformes logicielles (par exemple, Mac, Windows, Android, iOS, etc) mais également sur plusieurs plateformes matérielles tel que des centres de données mainframe aux smartphones.

3.3 Présentation de l'extracteur des métriques CK(Chidamber-Kemerer metrics)

Dans cette partie nous utilisons principalement deux outils à savoir :

- L'outil CK(Chidamber-Kemerer metrics) pour l'extraction des métriques de codes
- Weka pour le système d'apprentissage automatique

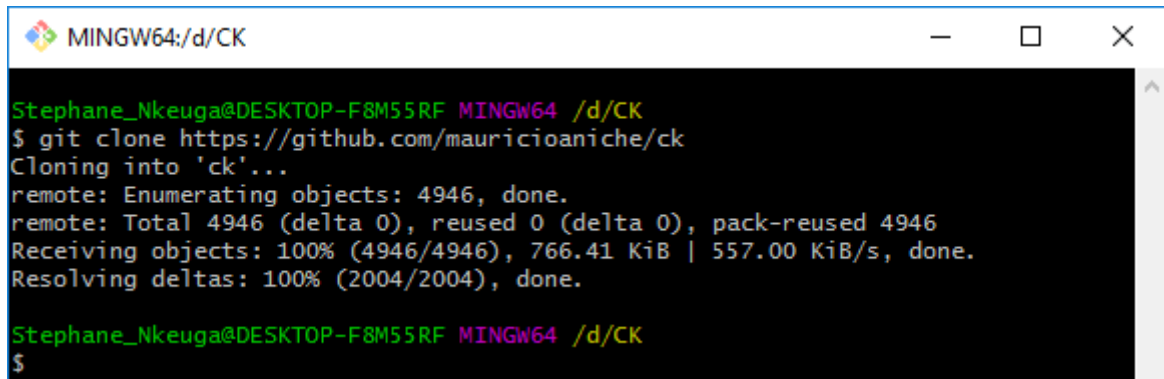
3.3.1 Fonctionnement CK(Chidamber-Kemerer metrics)

Pour l'extraction des métriques de code source nous nous sommes servis de la version 0.6.2 de l'outil CK(Chidamber-Kemerer metrics). CK est un outil écrit en langage Java par Mauricio Aniche Assistant Professor in Software Engineering at TU Delft au Pays Bas. CK calcule des mesures de code au niveau de la classe et au niveau de la métrique dans les projets Java au moyen d'une analyse statique (c'est-à-dire sans avoir besoin de code compilé).

3.3.2 Utilisation de CK(Chidamber-Kemerer metrics)

Son utilisation est assez simple, mais vous devez disposer au moins de la version 8 de java. Pour utiliser la dernière version vous devez cloner et générer le point jar avec les commandes suivantes :

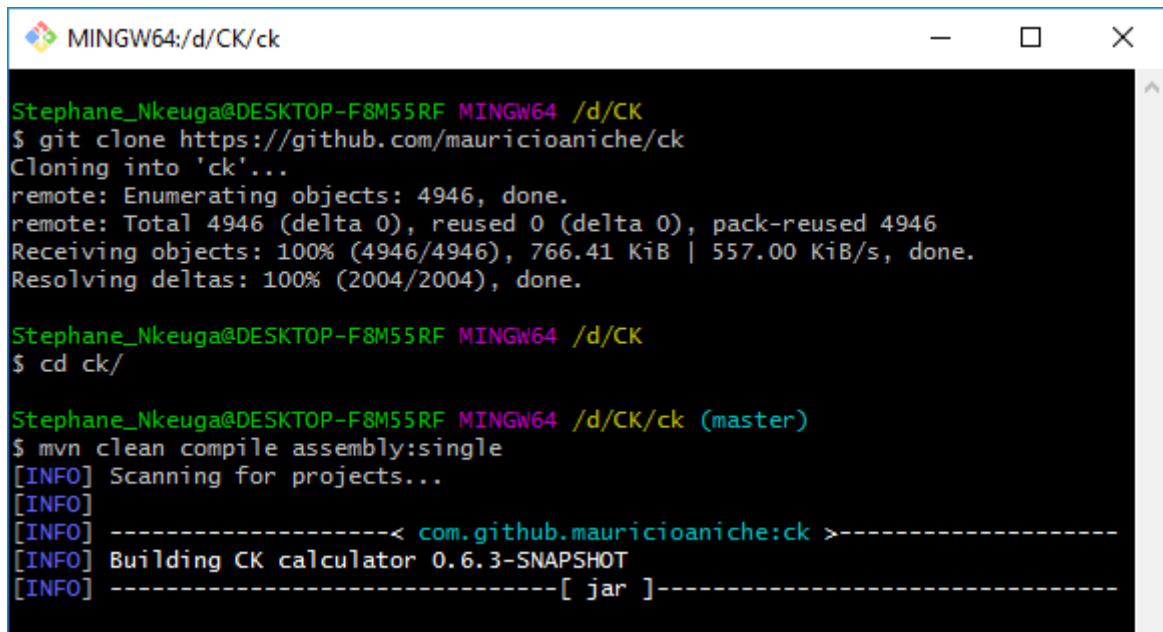
- **git clone <https://github.com/mauricioaniche/ck>** : permet de cloner l'outil sur son ordinateur comme l'illustre la figure suivante



```
MINGW64:/d/CK
Stephane_Nkeuga@DESKTOP-F8M55RF MINGW64 /d/CK
$ git clone https://github.com/mauricioaniche/ck
Cloning into 'ck'...
remote: Enumerating objects: 4946, done.
remote: Total 4946 (delta 0), reused 0 (delta 0), pack-reused 4946
Receiving objects: 100% (4946/4946), 766.41 KiB | 557.00 KiB/s, done.
Resolving deltas: 100% (2004/2004), done.
Stephane_Nkeuga@DESKTOP-F8M55RF MINGW64 /d/CK
$
```

FIGURE 3.1 – Clonage du l'outil CK

- **mvn clean compile assembly :single** : permet de generer le point jar comme l'illustre la figure suivante



```
MINGW64:/d/CK/ck
Stephane_Nkeuga@DESKTOP-F8M55RF MINGW64 /d/CK
$ git clone https://github.com/mauricioaniche/ck
Cloning into 'ck'...
remote: Enumerating objects: 4946, done.
remote: Total 4946 (delta 0), reused 0 (delta 0), pack-reused 4946
Receiving objects: 100% (4946/4946), 766.41 KiB | 557.00 KiB/s, done.
Resolving deltas: 100% (2004/2004), done.

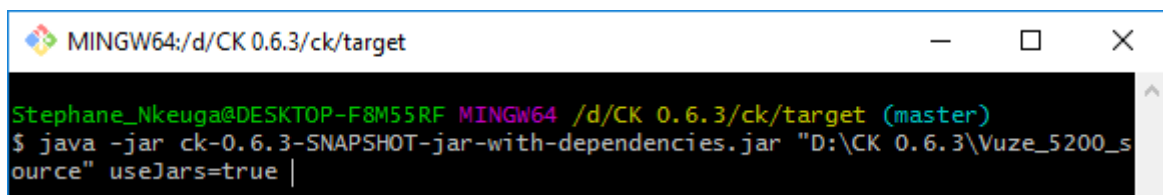
Stephane_Nkeuga@DESKTOP-F8M55RF MINGW64 /d/CK
$ cd ck/

Stephane_Nkeuga@DESKTOP-F8M55RF MINGW64 /d/CK/ck (master)
$ mvn clean compile assembly:single
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.github.mauricioaniche:ck >-----
[INFO] Building CK calculator 0.6.3-SNAPSHOT
[INFO] -----[ jar ]-----
```

FIGURE 3.2 – Générer le point jar

? **java -jar ck-x.x.x-SNAPSHOT-jar-with-dependencies.jar <project dirrectory> <use-Jars :true|false>**

Project directory fait référence au répertoire où CK peut trouver tout le code source à analyser. CK recherchera récursivement les fichiers .java. CK peut utiliser les dépendances du projet pour améliorer sa précision. Les paramètres useJars indiquent à CK de rechercher tous les fichiers .jar dans le répertoire et de les utiliser pour mieux résoudre les types. A la fin l'outil CK générera quatre fichiers csv : un fichier class.csv qui contiendra les métriques au niveau des classes, method.csv qui contiendra les métriques au niveau des méthodes , field.csv qui contiendra les métriques au niveau des champs et variable.csv qui contiendra les métriques au niveau des variables. Dans ce memoire nous atardons sur le fichiers class.csv



```
MINGW64:/d/CK 0.6.3/ck/target
Stephane_Nkeuga@DESKTOP-F8M55RF MINGW64 /d/CK 0.6.3/ck/target (master)
$ java -jar ck-0.6.3-SNAPSHOT-jar-with-dependencies.jar "D:\CK 0.6.3\Vuze_5200_s
ource" useJars=true |
```

FIGURE 3.3 – Calcul des métriques

3.4 WEKA(Waikato Environment for Knowledge Analysis)

Weka (acronyme pour Waikato environment for knowledge analysis, en français : « environnement Waikato pour l'analyse de connaissances ») est une suite de logiciels d'apprentissage

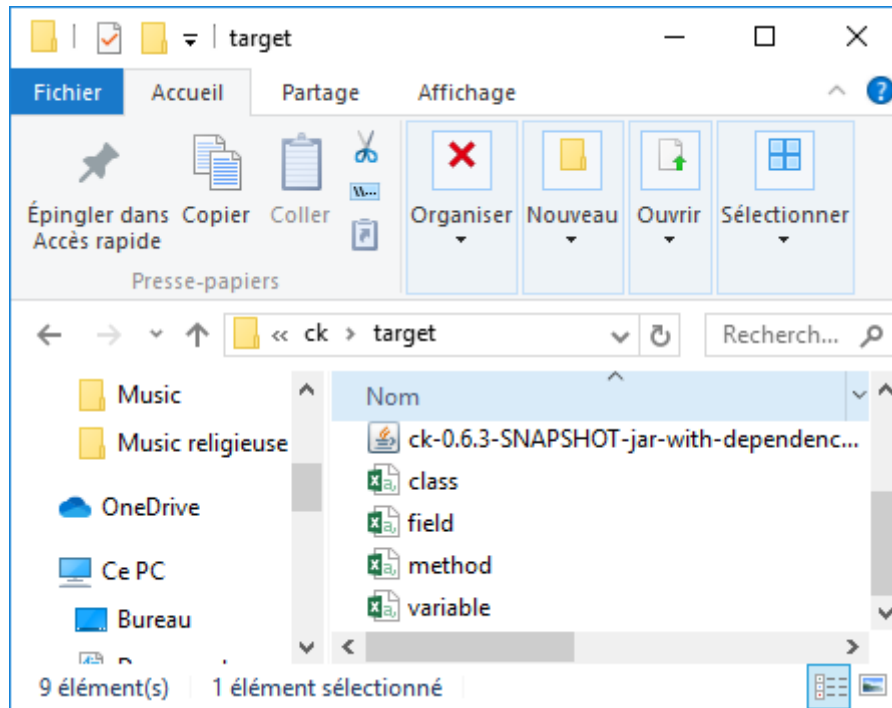


FIGURE 3.4 – Aperçu des fichiers

automatique écrite en Java et développée à l'université de Waikato en Nouvelle-Zélande. Weka est un logiciel libre disponible sous la Licence publique générale GNU (GPL). Weka est un logiciel open source fournissant des outils pour le prétraitement des données, Weka met en œuvre plusieurs algorithmes d'apprentissage automatique et des outils de visualisation permettent de développer des techniques d'apprentissage automatique et les appliquer à des problèmes d'exploration de données du monde réel. Ce que WEKA propose est résumé dans le schéma suivant

3.4.1 Présentation de WEKA

Le logiciel WEKA peut être utilisé sous à travers deux modes principaux. Le premier mode est une interface de commande en ligne **Simple CLI** qui est un interpréteur de ligne de commande. Le deuxième mode est une interface graphique **Explorer**. L'interface graphique du logiciel présente six onglets correspondant soit à des étapes du processus d'apprentissage, soit des classes d'algorithmes de classification (supervisée ou non) :

- **Preprocess** : La saisie des données, l'examen et la sélection des attributs, les transformations d'attributs.
- **Classify** : Les méthodes de classification.
- **Cluster** : Les méthodes de segmentation (clustering).
- **Associate** : Les règles d'association.
- **Select attributes** : L'étude et la recherche de corrélations entre attributs.
- **Visualize** : représentations graphiques des données.

Après l'installation de WEKA nous avons l'interface suivante :

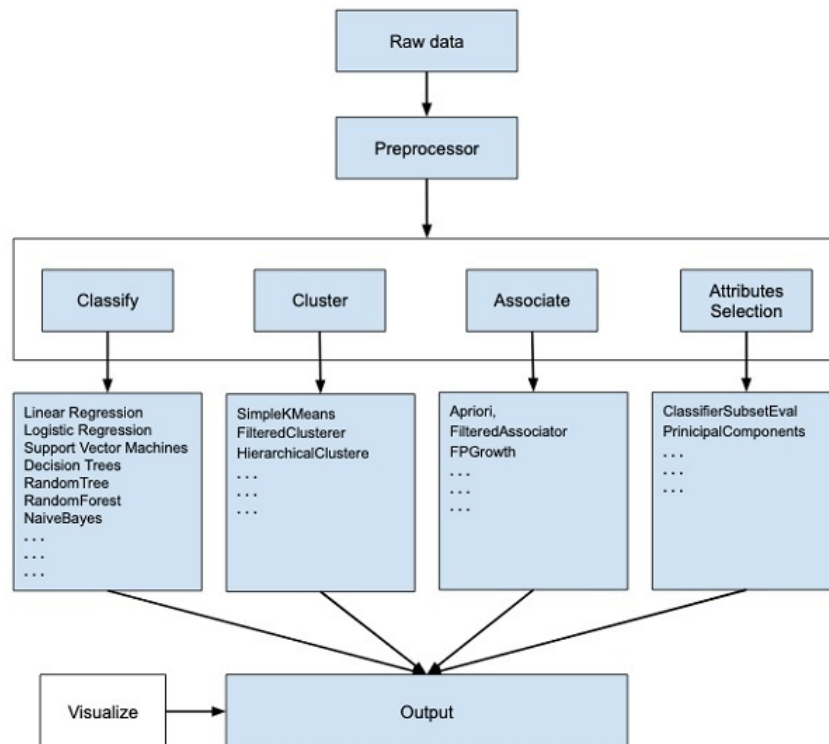


FIGURE 3.5 – Structure du logiciel Weka

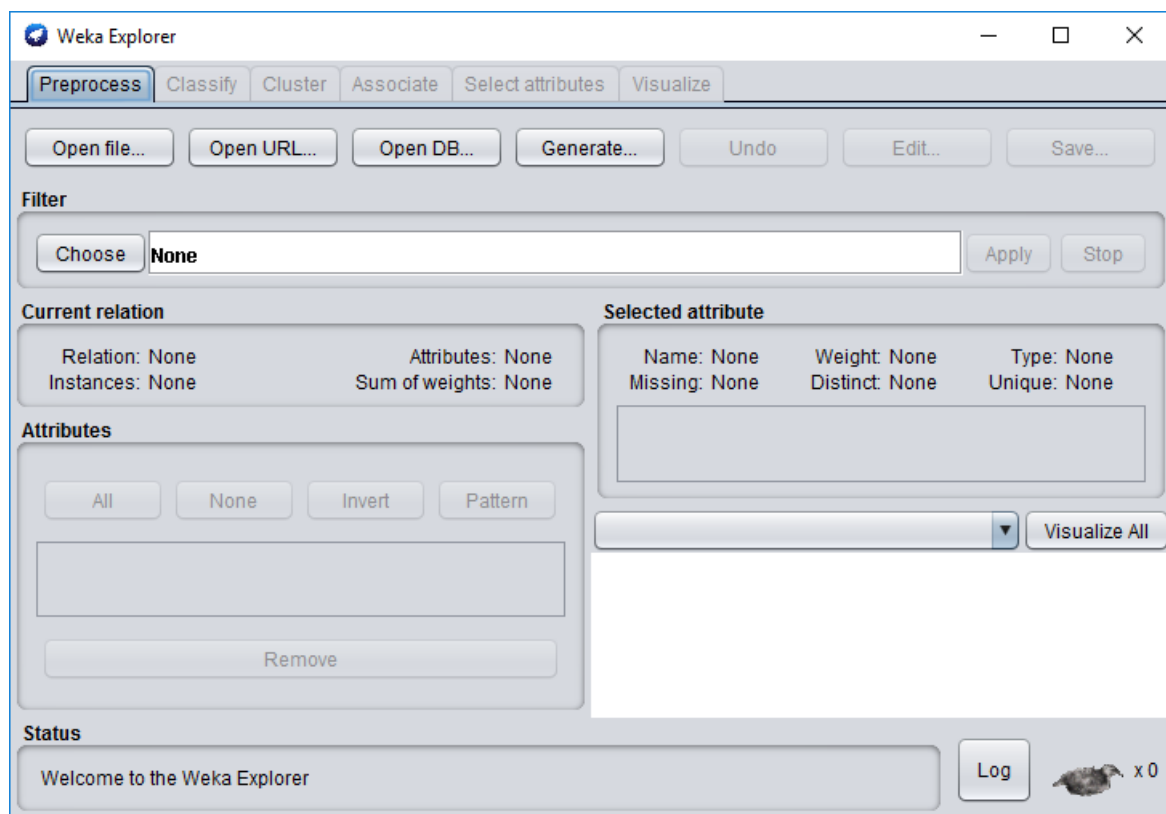


FIGURE 3.6 – Interface graphique de WEKA

3.4.1.1 Onglet Preprocess

Les données sont de plus en plus volumineuses dans l'industrie et le traitement de données constitue un véritable défi pour les systèmes informatiques. Le logiciel Weka se présente comme une solution efficace pour le traitement de données même celles de grandes envergures appelées « Big Data ». Pour être traitées, les données doivent être entrées sous les formats ARFF, CSV, Binaire, BDD, SQL et URL. Le format le plus utilisé est le format ARFF. Les données sous ces formats sont compatibles si elles sont bien structurées. La structure des données se compose de noms des données(@relation) suivis des attributs (@attribut) suivis de la variable de classe à prédiction (@data). Les données peuvent contenir divers types numérique continue, numérique discrète, catégorie, avec ou sans relation d'ordre (par ex. : rouge/vert/bleu), binaire (vrai/faux), les données structurées : arbres, graphes. Les attributs sont des réel (real), des chaînes de caractères (string) et des dates (date). Le principal outil de prétraitement de données est le filtre. Il existe deux types de filtres avec Weka : l'un non-supervisé et l'autre supervisé. Il existe d'autres types de filtre pour les attributs et pour les exemples :

- **Attribute Selection Filter** : sélection d'attributs selon les classes, par exemple, gain en information ;
- **Discretize Filter** : discrétise un intervalle d'attributs numériques vers des attributs nominaux ;
- **Nominal To Binary Filter** : Conversion d'attributs nominaux vers binaires ;
- **Numeric Transformation Filter** : Transformation d'attributs numériques selon une méthode à préciser (Racine carrée, val. Absolue).

3.4.1.2 Onglet Classify

Les classificateurs sont des modèles qui permettent à partir des modèles entrés de prédire les valeurs nominales numériques. Ces outils permettent d'obtenir une valeur chiffrée résultante des données et expériences à partir d'algorithme qui selon leur performance répondent à des indications précises sur l'attribut étudié.

Ils existent plusieurs algorithmes de classification dans le logiciel Weka. Les listes ne sont pas exhaustives pour ne citer que la régression linéaire, la régression logistique, les machines à vecteurs de soutien, les arbres de décision, RandomTree, RandomForest, NaiveBayes, etc

3.4.1.3 Onglet Cluster

Un algorithme de regroupement permet de trouver des groupes d'instances similaires dans l'ensemble des données. WEKA prend en charge plusieurs algorithmes de regroupement tels que EM, FilteredClusterer, HierarchicalClusterer, SimpleKMeans.

3.4.1.4 Onglet Associate

Pour montrer le contenu de l'onglet association nous faisons une remarque selon laquelle on a constaté que les gens qui achètent de la bière achètent des couches en même temps. Bien que cela ne semble pas très convaincant, cette règle d'association a été extraite d'énormes bases de données de supermarchés. Trouver de telles associations devient vital pour les supermarchés où l'on stocke les couches à côté de celles que les clients peuvent acheter, et où le chiffre

d'affaires est en augmentation. Cet onglet contient des schémas pour les règles d'association d'apprentissage, et les apprenants sont choisis et configurés selon les attentes voulues.

3.4.1.5 Onglet Sélection des attributs

Lorsque la base de données contient un grand nombre d'attributs, il y en aura plusieurs qui ne seront pas significatifs pour l'analyse. Ainsi, la suppression des attributs indésirables de l'ensemble de données devient une tâche importante dans l'élaboration d'un bon modèle d'apprentissage automatique. Weka propose d'examiner visuellement les données et décider de leurs attributs pertinents, ce qui pourrait être une tâche énorme pour les bases de données contenant un grand nombre d'attributs. Heureusement, WEKA propose plusieurs algorithmes de sélection automatique des caractéristiques tel que ClassifierSubsetEval et PrincipalComponents. En résumé la sélection des attributs consiste à rechercher toutes les combinaisons possibles d'attributs dans les données pour trouver le sous-ensemble d'attributs qui convient le mieux à la prédiction.

3.4.1.6 Onglet Visualisation

L'option Visualiser permet de visualiser les données traitées pour les analyser

CONCLUSION GÉNÉRALE

BIBLIOGRAPHIE

- [1] Stuart J Russell and Peter Norvig. *Artificial intelligence : a modern approach*. Malaysia ; Pearson Education Limited,, 2016.
- [2] Philip B Crosby. *Quality is free : The art of making quality certain*, volume 94. McGraw-hill New York, 1979.
- [3] Maurice Howard Halstead et al. *Elements of software science*, volume 7. Elsevier New York, 1977.
- [4] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4) :308–320, 1976.
- [5] Thomas J McCabe and Charles W Butler. Design complexity measurement and testing. *Communications of the ACM*, 32(12) :1415–1425, 1989.
- [6] ISO/IEC. Iso/iec 9126-3 software engineering -product quality- part 3 : Internal metrics. 2003.
- [7] James M Bieman. Metric development for object-oriented software. *Software Measurement : Understanding Software Engineering*, pages 75–92, 1996.
- [8] Alexander Serebrenik and Mark van den Brand. Theil index for aggregation of software metrics values. In *2010 IEEE International Conference on Software Maintenance*, pages 1–9. IEEE, 2010.
- [9] Bogdan Vasilescu, Alexander Serebrenik, and Mark van den Brand. Comparative study of software metrics? aggregation techniques. *Proceedings of the International Workshop Benevol*, 2010, 2010.
- [10] Gary Bradski, Adrian Kaehler, and Vadim Pisarevsky. Learning-based computer vision with intel’s open source computer vision library. *Intel technology journal*, 9(2), 2005.
- [11] Philip K Chan and Salvatore J Stolfo. Toward scalable learning with non-uniform class and cost distributions : A case study in credit card fraud detection. In *KDD*, volume 1998, pages 164–168, 1998.
- [12] Xuedong D Huang, William H Gates, Eric J Horvitz, Joshua T Goodman, Bradley A Brunell, Susan T Dumais, Gary W Flake, Trenholme J Griffin, and Oliver Hurst-Hiller. Web-based targeted advertising in a brick-and-mortar retail establishment using online customer information, January 3 2008. US Patent App. 11/427,764.
- [13] Igor Kononenko. Machine learning for medical diagnosis : history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1) :89–109, 2001.
- [14] Bernard Marr. A short history of machine learning—every manager should read. *Forbes*. <http://tinyurl.com/gslvr6k>, 2016.
- [15] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3) :103–130, 1997.

- [16] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.
- [17] Klaus Hechenbichler and Klaus Schliep. Weighted k-nearest-neighbor techniques and ordinal classification. 2004.
- [18] Ted Hong and Dimitris Tsamis. Use of knn for the netflix prize. *CS229 Projects*, 2006.
- [19] CE-Weaver Shannon. W. :(1949) the mathematical theory of communication. *Press UoI, editor*, 1948.
- [20] Norman E Fenton and Martin Neil. A critique of software defect prediction models. *IEEE Transactions on software engineering*, 25(5) :675–689, 1999.
- [21] Boehm Barry et al. Software engineering economics. *New York*, 197, 1981.
- [22] Bradford Clark, Sunita Devnani-Chulani, and Barry Boehm. Calibrating the cocomo ii post-architecture model. In *Proceedings of the 20th international conference on Software engineering*, pages 477–480. IEEE, 1998.