**1. Introduction**

Les métriques présentées dans cet article sont des métriques de test orientées objet. Le but de cet article n'est pas de mentionner toutes les métriques existantes ou de présenter entièrement les métriques mentionnées, mais de sensibiliser le lecteur à leur existence et d'offrir des références pour des lectures complémentaires. L'objectif de ce travail est de présenter une large revue de la littérature existante sur les mesures de test OO, L'objectif des métriques examinées proposées pour les systèmes OO est de se concentrer sur les métriques de produit pouvant être appliquées à une conception avancée ou au code. Plusieurs auteurs ont formulé leurs suggestions qui doivent être prises en compte lors de la définition des métriques pour les logiciels. Les métriques doivent être définies en poursuivant un objectif clair et leur calcul doit être facile, il est préférable que leur extraction soit automatisée par un outil. L'objectif de ce travail est de fournir aux chercheurs un aperçu de l'état actuel des métriques pour la programmation orientée objet, en mettant l'accent sur les forces et les faiblesses de chaque proposition existante. Ainsi, les chercheurs peuvent avoir une vue d'ensemble approfondie du travail déjà accompli et de celui qui reste à réaliser dans le domaine des métriques de test pour la programmation orientée objet.

**2. Metrics de Test Object-Oriented**

Nous allons maintenant présenter les propositions de métriques sélectionnées pour considération et qui peuvent mieux illustrer le contexte actuel des métriques de test pour la programmation orientée objet.

**Nombre de méthodes surchargées (NORM).**

Elle permet de calculer le nombre de méthodes

surchargées, pour une classe donnée.

**Méthode de calcul:**

.c: une classe,

.SURc: l’ensemble des méthodes surchargées de c.

La valeur de la métrique NORM pour la classe c sera donnée par la cardinalité de l’ensemble SURc.

Ce calcul est défini par l’équation suivante :

**Impact :**

- Un NORM élevé peut signaler une complexité accrue et une maintenance difficile, tandis qu'un NORM bas peut indiquer une conception plus simple mais potentiellement moins flexible.

**2Nombre de méthodes héritées (NMI).**

Cette métrique mesure le nombre de méthodes héritées par une sous-classe. Aucune mention n'est faite quant à savoir si cette héritage est public ou privé. Dans un langage tel que C++, nous devons considérer la possibilité que l'héritage soit privé. Ainsi, toute classe utilisant des méthodes d'une sous-classe n'aurait pas nécessairement accès à toutes les méthodes héritées.

**Méthode de calcul:**

.c: une classe, C: superclasse de c.

.METc: l’ensemble des méthodes de c.

.METC: l’ensemble des méthodes de C.

La valeur de la métrique NMI pour la classe c sera donnée par la cardinalité de l’intersection de l’ensemble METc et METC

Ce calcul est défini par l’équation suivante :

**Impact :**

-Un NMI élevé peut indiquer une forte dépendance sur les méthodes héritées, ce qui peut rendre le code plus complexe et difficile à maintenir (encapssulation fragile dans le cas d’heritage privee).

-Un NMI élevé peut également indiquer une bonne réutilisabilité du code, car les méthodes héritées peuvent être utilisées dans plusieurs parties du programme.

**Nombre de méthodes ajoutées à l'héritage (NMA).**

Une méthode est définie comme une méthode ajoutée dans une sous-classe s'il n'existe aucune méthode du même nom dans aucune de ses superclasses.

**Méthode de calcul:**

.c: une classe, C: superclasse de c.

.METc: l’ensemble des méthodes de c.

.METC: l’ensemble des méthodes de C.

La valeur de la métrique NMA pour la classe c sera donnée par la cardinalité de la difference entre METc et l’intersection de l’ensemble METc et METC

Ce calcul est défini par l’équation suivante :

**Impact :**

- Un NMI élevé peut contribuer à une meilleure spécialisation et à une extension des fonctionnalités, mais il peut également augmenter la complexité et les risques associés à la modification et à la maintenance du code.

**Indice de spécialisation (SIX). [Lorenz and Kidd, 1994]**

L'indice de spécialisation mesure dans quelle mesure les sous-classes remplacent le comportement de leurs superclasses.

.c: une classe, H(c): la position de c dans l’hiérarchie

d’heritage.

.METc: l’ensemble des méthodes de c.

.NORM(c): nombre de methode surhcage de c.

Il est défini par l’équation suivante :

**Impact :**

-Le SIX peut indiquer qu'il y a un excès de méthodes surchargées, ce qui pourrait signifier que l'abstraction initiale n'était peut-être pas appropriée. (qualité de la sous-classification)

-L&K suggèrent une valeur de 15% pour aider à identifier les classes pour lesquelles une action corrective pourrait être appropriée.

**Method Inheritance Factor (MIF):**

Définition : c'est une métrique de niveau système qui est le quotient de la somme des méthodes hérité dans chaque classe du système sur la somme des méthodes total (déclaré + hérité) de chaque classe du système.

**Formule :** tel que :

**Number of Public Methods Defined (PMd):**

Définition: c'est une métrique de niveau classe qui retourne la somme de toutes les méthodes publiques déclaré par une classe donnée.

**NPMi (Number of Public Methods inherited) :**

Définition: c'est une métrique de niveau héritage qui retourne la somme de toutes les méthodes publiques hérite par la classe parent d'une classe donnée.

**PMR (Public Methods Ratio) :**

Définition: c'est une métrique de niveau classe qui retourne le quotient du nombre de méthodes publique sur le nombre de méthode globale de la classe

Formule :

**IMR (Inherited Methods Ratio) :**

Définition: c'est une métrique de niveau classe qui retourne le quotient du nombre de méthodes hérité sur le nombre de méthode globale de la classe

**Formule :**

**MHF (Method Hidden Factor) :**

Définition: c'est une métrique de niveau système qui est le quotient de la somme des méthodes cachée dans chaque classe du système sur la somme des méthodes déclaré de chaque classe du système. Une méthode chachée est une méthode privée.

* 1. **Formule :** tel que :

**NHMd (Number of Hidden Methods defined) :**

Définition: c'est une métrique de niveau classe qui retourne la somme de toutes les méthodes cachées déclaré par une classe donnée.

**NHMi (Number of Hidden Methods inherited) :**

Définition: c'est une métrique de niveau héritage qui retourne la somme de toutes la méthode cachée hérite par la classe parent d'une classe donnée.Coupling factor:

Définition: CF est défini comme le rapport entre le nombre maximal possible de couplages dans le système et le nombre réel de couplages non imputables à l'héritage. Autrement dit, cette métrique compte le nombre de communications entre classes.

**Formule**:

**où** :

La relation client-fournisseur (Cc => Cs) signifie que la classe cliente, Cc, contient au moins une référence non héritée à une fonctionnalité de la classe fournisseur, Cs.

TC = Nombre total de classes dans le système en question.

**Polymorphism Factor :**

Définition:

PF est défini comme le rapport entre le nombre réel de situations polymorphiques différentes possibles pour la classe Ci et le nombre maximum de situations polymorphiques distinctes possibles pour la classe Ci.

PF est le nombre de méthodes qui redéfinissent des méthodes héritées, divisé par le nombre maximum de situations polymorphiques distinctes possibles. Top of Form

**Formule :** PF =

**où** :

Et :

DC (

TC = Nombre total de classes dans le système en question.

**Attribute Inheritance Factor (AIF) :**

Définition:

AIF est défini comme le rapport de la somme des attributs hérités dans toutes les classes du système en question au nombre total d'attributs disponibles (définis localement plus hérités) pour toutes les classes.

**Formule :**

**où :**

**Et :**

= le nombre d'attributs déclarés dans une classe

= le nombre d'attributs pouvant être invoqués en association avec

= le nombre d'attributs hérités (et non redéfinis dans .

**TC** = Nombre total de classes dans le système en ques

**Method Hidden Factor (MHF) :**

Définition :

MHF est défini comme le rapport de la somme des invisibilités de toutes les méthodes définies dans toutes les classes au nombre total de méthodes définies dans le système en question.

L'invisibilité d'une méthode est le pourcentage des classes totales à partir desquelles cette méthode n'est pas visible.

En d'autres termes, MHF est le rapport des méthodes cachées - méthodes protégées ou privées - au total des méthode

Formule : MHF =

**où**  est le nombre de méthodes déclarées dans une classe, et est le nombre de méthodes déclarées dans une classe, et

**Et :**

is\_visible(

TC = Nombre total de classes dans le système en question.

Dans des langages où il existe le concept de méthode protégée, la méthode est comptée comme une fraction entre 0 et 1 :

**Attribute Hiding Factor :**

Définition :

AIF est défini comme le rapport de la somme des attributs hérités dans toutes les classes du système en question au nombre total d'attributs disponibles (définis localement plus hérités) pour toutes les classes.

**Formule :** AIF =

**où**

**et :**

: le nombre d'attributs déclarés dans une classe.

 : le nombre d'attributs pouvant être invoqués en association avec Ci.

 : le nombre d'attributs hérités (et non redéfinis dans Ci).

TC : le nombre total de classes dans le système en question.

**CYCLOMATIC COMPLEXITY “CC”:**

Cyclomatic Complexity est un logiciel procédural (appelé également fonction) populaire

métrique égale au nombre de décisions pouvant être prises dans une procédure [MCC76].

Une décision est définie comme une occurrence de mots-clés tels que : « while », « for », « foreach »,"continue", "if", "case", "goto", "try" et "catch" dans la fonction. La complexité cyclomatique est la somme de ces constructions.

Formules : nombes d'arcs - nombres des sommets + 2 = CC

**WEIGHTED METHODS PER CLASS “WMC”:**

La métrique Méthodes pondérées par classe est une somme de complexités de méthodes définies

dans une classe. Il représente la complexité d'une classe dans son ensemble et peut être utilisé pour

indiquer l’effort de développement et de maintenance pour une classe.

Formules : WMC = ∑ Ci(somme de complexités).

**LACK OF COHESION METHODS “LCOM”:**

La cohésion d'une classe est caractérisée par l'étroitesse des relations entre les méthodes locales et les attributs locaux. Le manque de cohésion (LCOM) mesure la dissemblance des méthodes dans une classe par variable d'instance ou attributs. Un module hautement cohérent doit être autonome, une cohésion élevée indique une bonne subdivision des classes.

Formule : LCOM =| P | - | Q |, si | P | > | Q |

= 0 sinon

P = {( I i , I j ) | I i ∩ I j = 0} et Q = {((I i , I j ) | I i ∩ I j ≠ 0} . Si tous les n ensembles {( I1},.........(In )} valent 0 alors P=0.

Soit (Ix) = ensemble de toutes les variables d'instance utilisées par la méthode Mi.

**AFFERENT – EFFERENT COUPLING “CA-CE”:**

Couplage afférent : nombre d'entités externes qui dépendent de l'entité courante.

Couplage efférent : nombre d'entités externes dont dépend l'entité courante.

Formules :Pour calculer le **Ca** pour un package, comptez le nombre de classes du package qui ont des dépendances. Vous pouvez procéder une par une pour toutes les classes de ce package, puis procéder à l'union des classes dépendantes.Pour calculer le Ce pour un package, comptez le nombre de classes dépendant d'autres packages à l'intérieur du package analyséemballer. Comptez pour toutes les classes de ce package, puis prenez le syndicat.

**INSTABILITY “I”:**

Le rapport du couplage efférent (Ce) au couplage total (Ce + Ca) tel que I = Ce / (Ce + Ca). Cette mesure est un indicateur de la résilience du paquet au changement. La plage de cette métrique est comprise entre 0 et 1, I=0 indiquant un package complètement stable et I=1 indiquant un package complètement instable.

Formules :I = Ce / (Ce + Ca)

**ABSTRACTNESS “A” :**

Le rapport entre le nombre de classes abstraites (et d'interfaces) dans le package analysé et le nombre total de classes dans le package analysé. La plage de cette métrique est comprise entre 0 et 1, A=0 indiquant un package complètement concret et A=1 indiquant un package complètement abstrait.

**DISTANCE « D » :**

la distance D indique à quelle distance un package se trouve de la séquence principale.

Formule : D = A + I -1

**Number of attributes**

Description:

Cette métrique compte simplement le nombre total d'attributs (également appelés variables d'instance ou champs) au sein d

'une classe. Les attributs représentent l'état d'un objet et définissent ses caractéristiques ou propriétés.

**Number of static attributes**

Description:

Cette métrique compte le nombre total d'attributs statiques définis dans une classe. Les attributs statiques appartiennen

t à la classe elle-même et sont partagés entre toutes les instances de la classe.

**Number of public attributes**

Description:

Cette métrique compte le nombre total d'attributs publics définis dans une classe. Les attributs publics peuvent être acc

essibles depuis l'extérieur de la classe et sont déclarés avec le modificateur d'accès public.

**Number of methods**

Description:

Cette métrique compte le nombre total de méthodes (y compris les méthodes d'instance et les méthodes statiques) définies

dans une classe. Les méthodes définissent le comportement de la classe et peuvent manipuler l'état de l'objet ainsi que r

éaliser diverses tâches.

**Number of static methods**

Description:

Cette métrique compte le nombre total de méthodes statiques définies dans une classe. Les méthodes statiques appartiennen

t à la classe elle-même et peuvent être appelées directement sur la classe sans nécessiter la création d'une instance.

**Number of classes**

Description:

Cette métrique compte le nombre total de classes dans un système logiciel ou dans un module spécifique. Les classes serve

nt de modèles pour créer des objets, encapsuler des données et définir des méthodes pour opérer sur ces données.

**3. Conclusions et travaux futurs.**

Le document présente une suite métrique de base pour le Test orientée objet. Les données métriques offrent une rétroaction rapide aux concepteurs de logiciels et aux gestionnaires. L'analyse et la collecte de ces données peuvent prédire la qualité de la conception. Si elles sont utilisées de manière appropriée, elles peuvent conduire à une réduction significative des coûts de mise en œuvre globale et à des améliorations de la qualité du produit final. La qualité améliorée, à son tour, réduit les efforts de maintenance future. Utiliser des indicateurs de qualité précoce basés sur des preuves empiriques objectives est donc un objectif réaliste.

À l'avenir, nous concevrons un nouvel ensemble de métriques permettant de Tester nos programmes.

***References***

1.An Overview of Object-Oriented Design Metrics , Daniel Rodriguez et Rachel Harrison, RUCS/2001/TR/A , Mars 2001.

2.International Journal of Information Technology & Systems, Vol. 2; No. 1: ISSN: 2277-982

3.Applying and Interpreting Object Oriented Metrics Dr. Linda H. Rosenberg

4.https://en.wikipedia.org/wiki/Software\_package\_metrics

5.METRICS FOR OBJECT ORIENTED DESIGN (MOOD) TO ASSESS JAVA PROGRAMS, Prof. JUBAIR J. AL-JA'AFERKHAIR EDDIN M. SABRI, University of Jordan