UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ANALYSE DE L'ÉVOLUTION DES CONCEPTS DE COHÉSION ET DE COUPLAGE : 1979-2008

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

WIDAD CHAMI

DÉCEMBRE 2008

À mes très chers parents

À mes sœurs bien aimées Leïla et Aïda

REMERCIEMENTS

Je tiens tout d'abord à remercier chaleureusement mon directeur de recherche, Monsieur Ivan Maffezzini, et lui adresser ma plus profonde gratitude pour l'apprentissage qu'il m'a permis en acceptant travailler avec moi. Je le remercie, pour son appui, sa patience illimitée, sa disponibilité et sa gentillesse.

J'aimerais remercier Madame Rim Bejaoui, Madame Aida Chami, d'avoir aidé à rendre ce mémoire clair.

Je remercie aussi les professeurs qui ont accepté de corriger mon mémoire.

Enfin, tous mes remerciements s'adressent à ma famille, qui m'a supportée tout au long de ce projet de recherche, en particulier mes parents, mes sœurs et mes frères.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES

Acronyme	Signification
LCC	Loose Class Cohesion
CBMC	Cohesion Based on Member Connectivity
PC	Composant principal
COO	Conception Orientée Objet
C&C	Cohésion et couplage
CVL	Cycle de vie du logiciel
DLC	Design-Level Cohesion
DFC	Design-Functional Cohesion
EIA	Electronic Industries Association
Fc	Facteur de connectivité
Fs	Facteur de structure
FC	Functional Cohesion
GL	Génie logiciel
Gr	Graphe de référence
IODG	Input/Output Dependence Graph
IEEE	Institute of Electrical and Electronics Engineers
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
LCOM	Lack of Cohesion in Methods
LC	Loose Cohesiveness
MC	Module Cohesiveness
MCC	Most Cohesive Component
OO	Orienté Objet
PCA	Principal Component Analysis
PFG	Processing element Flow Graph
RFC	Response For a Class
SWEBOK	Software Engineering Body of Knowledge
TCC	Tight Class Cohesion
TC	Tight Cohesiveness
TLFI	Trésor de la langue française informatisé

UML	Unified Modelling Language

RÉSUMÉ

Depuis leur apparition, la cohésion et le couplage (C&C) ont fait couler beaucoup d'encre, et ce en raison du rôle qui leur a été attribué dans la détermination des traits de plus qu'un attribut de la qualité du logiciel. Cette détermination a pris plusieurs aspects modelés selon les besoins et les réalisations de chaque étape d'évolution par lesquels ils sont passés. Ces aspects ont contribué grandement au retrait d'une large ambigüité qui a couvert le couple C&C comme tout autre concept mesurable de la qualité logicielle. En revanche, ils ont pris part à l'exacerbation du nombre de questions de recherche —quelques unes vagues et d'autres précises, à propos de ce qui a formé une preuve évidente de la difficulté d'utilisation de ces deux concepts. Depuis toujours la source du problème de cette difficulté est connue : définitions informelles, et la solution est unique et unanime : définition de mesures objectives et normalisées. Mais, les visions, qui diffèrent à cause des multiples chemins possibles à prendre et des divers moyens envisageables, rendent en quelque sorte la réalisation de cette solution difficile.

Dans notre étude nous avons analysé l'histoire de ces deux concepts en prenant comme point de départ le livre *Structured Design* de Yourdon et Constantine (1979). Nous avons ensuite suivi à la trace les changements de ces concepts tout au long des trois dernières décennies, en analysant une vingtaine d'articles marquants. Avec cette étude, nous avons découvert, qu'avec leurs visions différentes, les chercheurs sont arrivés à résoudre certains sous-problèmes (qui découlent du problème principal qui s'est avéré très complexe), tout en découvrant d'autres sous-problèmes qui ont besoin d'être solutionnés, pour que le tout permette un éclaircissement et une précision qui accordent une utilisation efficace du couple C&C, dans la production d'applications d'envergure où l'on applique les principes du génie logiciel (GL). Le résultat de notre étude peut être résumé par le « non » que nous avons donné comme réponse à notre question de recherche principale « Est-ce que les concepts de cohésion et de couplage ont des définitions assez précises pour pouvoir être employés de manière efficace dans le GL ? ».

Mots-clé : cohésion, complexité, couplage, modularisation, métrique, qualité.

INTRODUCTION

I.1 Introduction

La clarté et la précision conceptuelles sont les conditions essentielles de toute approche scientifique et technique à un problème. L'informatique et le génie logiciel (GL) n'échappent pas à cette règle. Même si ce sont des domaines encore relativement jeunes, même s'ils aspirent — comme ils le font ! — au niveau de succès des domaines techniques issus et alimentés par la science moderne, ils doivent mettre de l'ordre dans leur terminologie. Mais la clarté et la précision terminologiques demandent parfois des ralentissements théoriques, des pauses qui permettent la réflexion et des remises en question. Tout cela est loin d'être facile en informatique et en GL en raison du grand succès des ordinateurs qui, ayant envahi pratiquement tous les domaines imaginables, demandent toujours plus de ressources humaines consacrées à développer de nouvelles applications plutôt qu'à faire de l'ordre dans ce qui existe déjà.

Dès les débuts de l'informatique, deux approches se confrontent et se complètent : l'approche « ingénierie » et l'approche « mathématique », d'une part l'exigence de créer de nouvelles applications le plus vite possible et d'autre part celle d'établir des fondations théoriques. Dès l'instant où les applications sortent des laboratoires et deviennent des composantes essentielles à la gestion (banques, gouvernements, etc.) ou au contrôle des procédés (usines, centrales, etc.) naît la nécessité de trouver un compromis entre les approches formelles des mathématiciens et les approches pragmatiques des ingénieurs. C'est dans le cadre de cette recherche de compromis que, dans les années 1960, le ministère de la défense américain (le plus gros consommateur de logiciel) exige que l'informatique devienne « génie ». « Le génie logiciel naît parce qu'on a besoin d'une certaine assurance que, quand on achète un produit – si on a la chance de le recevoir ! –, il s'agit d'un produit de qualité qui exécute les fonctions demandées comme dans l'ingénierie "normale" » (Maffezzini, Martin et Premiana, 2005). Et l'ingénierie « normale » est fondée sur la maîtrise des processus pour réaliser des produits de qualité, au prix les plus bas possibles, dans les temps les plus courts possibles.

Ce n'est pas un hasard si, dans les mêmes années, D. Knuth, E. Dijkstra et C. A. R. Hoare pour ne citer que trois auteurs, posent les bases d'une approche scientifique à une informatique qui sort des laboratoires. C'est à cette même époque qu'a eu lieu la célèbre querelle des « goto » entre Knuth (défenseur de la lisibilité) (Knuth, 1977) et Dijkstra (défenseur du purisme de la programmation fondée sur le théorème de Boehm et Jacopini[1]) (Dijkstra, 1968). D'un certain point de vue, cette controverse est plus le symptôme d'un souci commun des deux auteurs que d'une différence profonde. Le souci commun est « un souci de maintenabilité » (Maffezzini, Martin et Premiana, 2005).

C'est également dans les mêmes années qu'un praticien comme E. Yourdon écrit avec L. Constantine un livre qui contribuera à former une génération d'informaticiens adeptes de la programmation structurée (Yourdon et Constantine, 1979). Et c'est surtout ce livre qui est à l'origine de la diffusion et du succès de deux mots « cohésion » et « couplage », qui depuis quarante ans, reviennent constamment dans la littérature informatique. Par exemple : qu'est-ce qui est aujourd'hui au centre des « services web », l'une des nouvelles approches à l'informatisation des entreprises, sinon le couplage ?

Nous mettrons un terme à cette introduction avec une mise au point terminologique. Nous sommes conscients des différences entre « informatique » et « génie logiciel » telles que présentées, par exemple, dans SWEBOK[2] (IEEE SWEBOK, 2001), mais aux fins de notre étude nous ferons abstraction de ces différences. Même si notre démarche est une démarche théorique fondée sur l'analyse de la littérature, notre mémoire a un aspect pratique car il vise à donner des indications utiles pour des projets ayant une certaine durée et une certaine complexité (des projets de « génie »). Le contexte nous fera parfois pencher vers le « génie logiciel » et parfois vers l'« informatique » mais, dans le cadre de notre étude, le lecteur peut considérer les deux termes comme étant presque synonymes.

I.2 Problèmatique

---

[1] Théorème qui dit que tout programme peut être écrit avec trois types d'instructions : séquence, itération et sélection. Cité en Randall et Tonies (1979).

[2] *Guide to the Software Engineering Body of Knowledge.*

Dans presque n'importe quel domaine, la qualité du produit final est extrêmement importante. Si importante que la satisfaction des besoins fonctionnels sans « qualité » n'est pratiquement d'aucune utilité pour le « succès » d'un produit. Mais la qualité est quelque chose d'extrêmement difficile sinon impossible à définir : dès que l'on analyse la qualité, et qu'on ne la considère pas comme un tout, on se retrouve souvent avec les mains vides (Schulmeyer, 1999). Et pourtant, dans le champ de la technique, on ne peut se contenter d'affirmer que la qualité est complètement subjective ou que la qualité est indéfinissable. Dans le champ de la technique, il convient de maîtriser la qualité si l'on veut que les produits réalisés aient la moindre probabilité de survie. Mais si on ne « sait pas » ce qu'elle est au juste, alors comment peut-on la contrôler ? Comment la vérifier ? Comment s'assurer de sa présence ? Si, quand les différents intervenants d'un même projet parlent de qualité, subsiste le risque qu'ils ne parlent pas de la même chose, comment imaginer que le produit final puisse être « de qualité » ?

Une des façons permettant d'aller au-delà de phrases terriblement ambiguës du genre : « il faut avoir un bon produit », « il doit être facile à entretenir », « il doit avoir une bonne disponibilité »[3], etc., est d'employer une approche de type « *dividi et impera* » (diviser pour régner). C'est ce que l'on a commencé à faire dès les années 1960. Pour « diviser » le problème de la qualité, on a introduit des descriptions assez rigoureuses (des modèles). Nous nous limiterons ici à citer les modèles de McCall[4] (1977), Boehm (1978) et celui de la série de normes 9126 d'ISO/IEC (ISO/IEC 9126-1, 2001 ; ISO/IEC 9126-2, 2003 ; ISO/IEC 9126-3, 2003 ; ISO/IEC 9126-4, 2004). Tous ces modèles, fondés sur un cadre assez formel, partent de l'idée que la qualité a une structure hiérarchique, dont les éléments de niveau inférieur influencent ceux du niveau supérieur. Ces modèles sont un premier pas pour faciliter la communication entre les différents intervenants.

Même si les termes employés sont parfois différents, les trois modèles que nous venons de citer sont fondés sur une structure hiérarchique à trois ou quatre niveaux. Pour faciliter la

---

[3] Comment ne pas souligner que ce sont surtout les « qualificatifs » qui font problème dans ces phrases !

[4] En fait, il s'agit du modèle de McCall, Richards et Walters (1977). Mais, afin d'alléger le texte, nous l'appellerons modèle de McCall (1977).

lecture, dans ce chapitre introductif, nous allons employer les termes de la norme ISO/IEC 9126. Dans cette norme, on décrit trois types de qualité des produits et chaque type est structuré en caractéristiques qui, à leur tour, peuvent être structurées en sous-caractéristiques[5]. La norme présente un total de dix (10) caractéristiques et vingt-huit (28) sous-caractéristiques. Considérons à titre d'exemple la définition de la sous-caractéristique interopérabilité : « *La capacité du produit logiciel d'interagir avec un ou plusieurs systèmes spécifiés* » (ISO/IEC 9126-1, 2001). Cette définition ne peut pas être employée pour caractériser la facette interopérabilité de la qualité d'un logiciel. Pour rendre l'interopérabilité « objectivement » vérifiable, la norme introduit le concept d'attribut qui se définit comme « *une propriété abstraite ou physique mesurable d'une entité* » (ISO/IEC 9126-1, 2001). C'est-à-dire que la norme introduit des entités mesurables pour rendre moins ambiguës les sous-caractéristiques qui ont, en général, plusieurs attributs.

Cette classification et les métriques associées aux attributs facilitent sans doute la gestion de la qualité même si un énorme problème reste ouvert : quelle relation existe-t-il entre les valeurs numériques des attributs et la « qualité attendue » ? Nous n'avons pas la prétention de pouvoir répondre à cette question mais, dans notre étude, nous espérons pouvoir apporter une modeste contribution en limitant énormément le champ d'investigation. Nous n'allons considérer que deux concepts : *cohésion* et *couplage* (C&C). Pourquoi ces deux concepts et pas d'autres ? Parce qu'il existe une littérature très riche à leur sujet et qu'ils restent des concepts importants pour la gestion de la qualité depuis les débuts du génie logiciel[6].

I.3 Objectifs et questions de recherche

Depuis la fin des années 1960 (c'est-à-dire depuis la naissance même du domaine du GL), les concepts de C&C sont parmi les concepts que l'on retrouve[7] le plus souvent dans la littérature. Mais, même si C&C font partie du langage quotidien des informaticiens[8], nous

---

[5] Un des types de la qualité (la qualité de fonctionnement) n'a pas de sous-caractéristiques.
[6] Et, bien sûr, parce que notre directeur de recherche nous a convaincu de leur importance.
[7] La liste des œuvres à citer serait ici bien trop longue. Qu'il suffise de dire que tous les livres de GL que nous avons consultés en parlent.
[8] Ou surtout parce qu'ils font partie du langage quotidien ?

avons l'impression que la signification de ces deux termes reste trop vague. Notre étude se propose donc d'analyser l'histoire de ces deux concepts pour mieux les comprendre et pour essayer de saisir leur évolution éventuelle. Loin de nous la prétention de donner les « bonnes » définitions de « cohésion » ou de « couplage » : nous nous proposons simplement d'étudier assez en détail l'histoire de ce couple de concepts pour que l'on soit un peu plus conscient de la complexité qui se cache derrière ces termes en apparence si simples. Tout cela non dans un but éminemment théorique, mais comme aide conceptuelle pour les praticiens qui sont quotidiennement confrontés à des systèmes « trop couplés » et à des modules à cohésion « trop faible ».

Notre hypothèse de base est que les concepts de C&C sont assez mal définis malgré le grand nombre de tentatives d'éclaircissement qui circulent depuis leur apparition. Pour tester notre hypothèse, nous posons la question de recherche suivante :

Est-ce que les concepts de cohésion et de couplage ont des définitions assez précises pour pouvoir être employés de manière efficace dans le GL ?

Pour répondre à cette question, nous essayerons de répondre aux sous-questions suivantes :

1. ***Depuis le début des années 1970, y a-t-il une évolution ou des évolutions des concepts de cohésion et de couplage vers une plus grande précision ?***
2. ***Est-ce qu'il existe des différences significatives dans la définition et dans l'emploi de ces termes?***
3. ***Les métriques pour la cohésion et le couplage permettent-elles de mieux éclaircir ces concepts ?***

La littérature présente C&C comme deux concepts clef de la qualité interne du logiciel. Le but principal de notre travail est de décortiquer ces deux concepts pour pouvoir évaluer s'ils « méritent » la position qu'on leur assigne. Pour ce faire, nous allons suivre à la trace sur trois décennies l'évolution (si évolution il y a) pour trouver et souligner les points de rupture et les changements de contexte dans l'emploi des termes.

Pour mieux scruter C&C, il s'agira :

- d'étudier les concepts de modularisation et de complexité ;
- d'étudier quelques modèles de qualité ;
- de mettre en relation C&C avec la complexité et avec la modularisation ;
- d'étudier quelques métriques

I.4 Démarche adoptée

Pour un mémoire fondé essentiellement sur l'étude de la littérature, plusieurs choix méthodologiques devaient être réglés :

1. Fallait-il se limiter à la production des dernières années ou couvrir l'évolution du couple C&C pratiquement depuis ses débuts ? Nous avons décidé de partir du début en regroupant les articles par décennies[9] pour mieux voir l'évolution.

2. Comment établir le début ? Le choix a été assez facile car pratiquement tous les auteurs s'accordent sur le livre « *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design* » de E. Yourdon et de L. Constantine (1979). Même s'il ne s'agit pas du premier ouvrage sur le sujet, il s'agit du plus pédagogique et de celui qui a influencé toute une génération de praticiens et de chercheurs.

3. Quel serait le nombre réaliste d'articles à analyser en détail ? Nous nous sommes fixés une limite inférieure de quinze articles à étudier en détail, plus un certain nombre d'autres articles à consulter pour nous aider à mieux faire le point sur la situation.

4. Comment choisir les articles parmi l'énorme quantité de publications des quarante dernières années ? Un premier tri a été effectué en partant des articles les plus cités dans les livres de génie logiciel que nous avions à notre disposition. Ensuite, nous avons intégré des articles cités dans les articles déjà choisis, en essayant de couvrir toujours plus d'aspects. *A posteriori*, nous avons l'impression que, même si certains choix ont été dictés seulement par nos goûts et ceux de notre directeur, l'échantillonnage choisi est assez représentatif de ce qui a été écrit sur C&C.

---

[9] Nous sommes conscients que la division en décennies n'est qu'une division d'ordre psychologique et que les éléments d'une décennie risquent de ne pas avoir une très grande cohésion. Mais cela nous a paru être la division la plus naturelle. Pour mettre l'évolution en évidence, nous aurions aussi pu diviser en « avant et après l'approche objet » mais, même si cette division a l'air plus objective, elle nous a semblé encore moins cohésive car, d'une part, l'entrée de l'approche objet couvre un laps de temps assez long et, de l'autre, même quand l'approche objet « domine », certaines études adoptant des approches structurées continuent à être produites.

I.5 Organisation du mémoire

Le chapitre 1, *Introduction*, présente la problématique, les objectifs et les questions de recherche, et enfin la démarche adoptée.

Le chapitre 2, *Origines des concepts de cohésion et couplage avec un excursus dans la « complexité »*, permet de mettre les concepts de cohésion et de couplage en contexte dès leur apparition.

Les chapitres 3, 4 et 5, respectivement *Cohésion et couplage : années 1980-1989*, *années 1990-1999* et *années 2000-2008*, comprennent une analyse individuelle des articles traitant le sujet C&C qui ont été sélectionnés pour chaque décennie, et une analyse globale de l'ensemble.

Le chapitre 6, *Conclusion* répond à nos questions de recherche, décrit les limites de notre étude et donne quelques indications sur notre apprentissage.

L'appendice A contient les transcriptions/résumés des articles que nous avons analysés en détail.

L'appendice B contient une liste de définitions « cohésion » et « couplage » que nous avons extraites de livres de génie logiciel avec quelques comparaisons.

L'appendice C contient des résumés d'articles traitant de la complexité.

I.6 Activités

La figure ci-dessous présente, sous forme de diagramme d'activités en UML, les activités principales que nous avons accomplies dans le cadre de notre étude. Nous donnons ici une brève description de chaque activité :

- *Fixation des objectifs et des questions de recherche.* Activité qui a été déclenchée par le choix du sujet. L'extrant de cette activité a été une première version du chapitre 1.

- *Lecture du livre de Yourdon et Constantine* (dont nous parlons au chapitre 3). Activité qui nous a permis de saisir la problématique de la cohésion et du couplage et les liens de ces concepts avec la qualité et en particulier la complexité. Nous avons ensuite étudié les modèles de qualité et la complexité cognitive (ces activités n'apparaissent pas dans la figure 1.1). L'extrant de cette activité a été le chapitre 3.

- *Choix des articles.* Suite aux deux activités précédentes, nous avons pu choisir les articles. Activité qui a été moins systématique que prévue, mais qui nous a forcé à glaner dans bien des textes avant de choisir.

- *Traduction/transcription/résumé.* Il s'agit de l'activité la plus longue et difficile car il s'agissait de comprendre des articles très différents qui demandaient souvent des connaissances mathématiques et en GL qui nous ont donc poussé à approfondir certains thèmes. Les sorties de cette activité sont les appendices.

- *Synthèse des articles.* En partant des sorties de l'activité précédente, il a fallu faire une synthèse un peu plus personnelle.

- *Commentaires articles et décennies.* Un bref commentaire/critique a été rédigé pour chaque article synthétisé et pour chaque décennie. Les chapitres 2, 4, 5, 6 ont été les sortie de cette activité et de la précédente.

- *Réponse aux questions*. Cette activité nous a permis de répondre aux questions posées lors de la première activité et de rédiger la conclusion.

- *Révision du mémoire*. Dernière activité avant le dépôt.

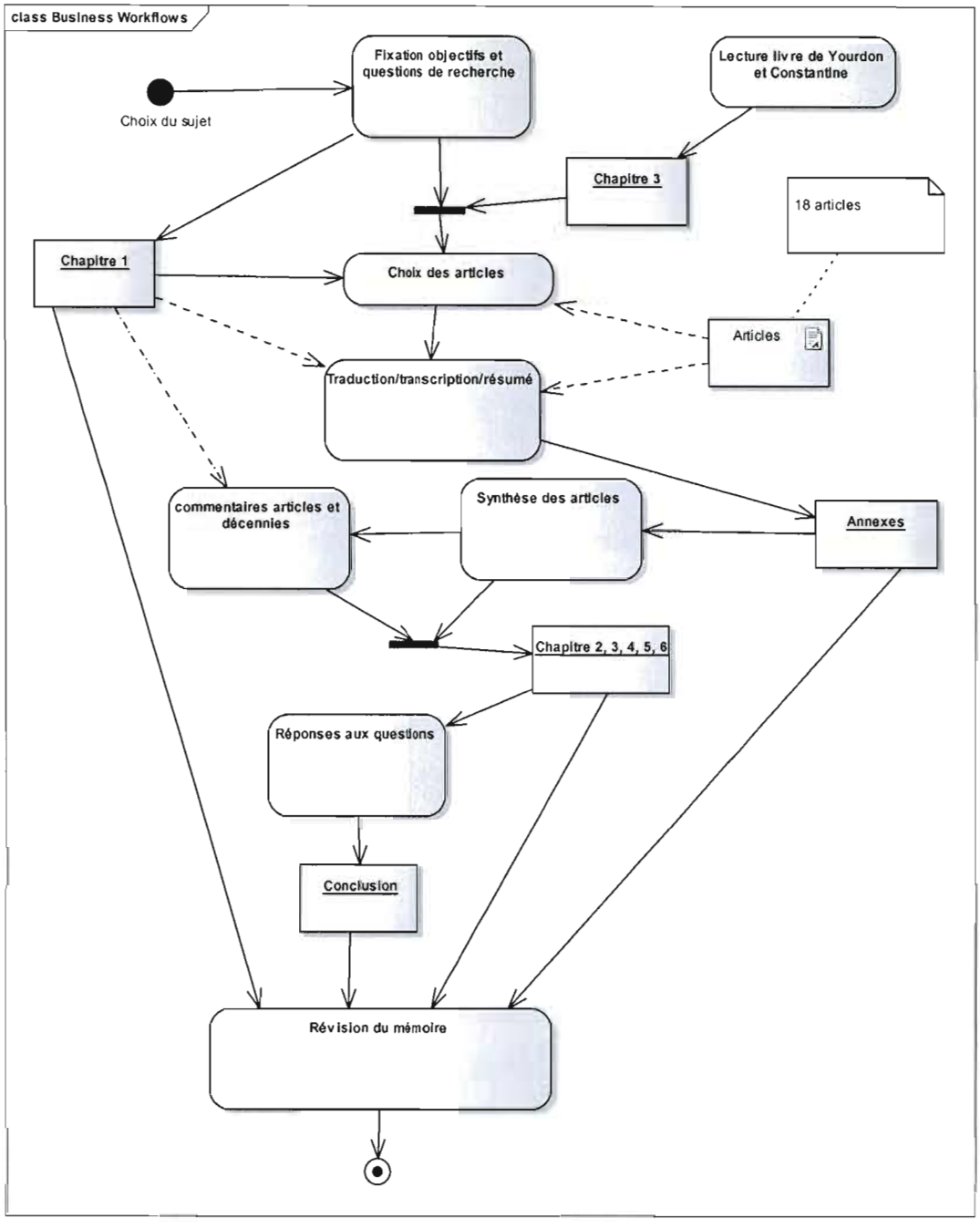


Figure I.1 Diagramme d'activités

CHAPITRE I

LE MODÈLE DE QUALITÉ D'ISO/IEC 9126-1

1.1 Introduction

Dans ce chapitre, après une brève présentation des modèles de qualité de McCall et de Boehm, qui constituent la base sur laquelle est fondé le modèle de qualité d'ISO/IEC 9126-1, nous étudions plus en détail ce dernier.

1.2 Modèle de qualité de McCall (1977)

Jim McCall est l'un des premiers qualiticiens du logiciel qui a essayé de faire passer la qualité d'un cadre qualitatif à un cadre où il aurait été plus facile de quantifier. Pour réaliser cela, il a proposé une décomposition et une structuration du concept de « qualité ».

« *McCall, dans son modèle, a essayé d'établir le lien entre les utilisateurs et les développeurs, en se concentrant sur un certain nombre de facteurs de qualité du logiciel qui reflètent les vues des utilisateurs et les priorités des réalisateurs* » (Milicic, 2005).

Ce modèle est sous forme d'une hiérarchie de quatre niveaux (voir figure ci-dessous) :

- Le premier niveau organise le cycle de vie du logiciel (CVL) en trois phases en fonction de l'emploi du logiciel :
    - *Opération* : l'emploi du produit ;
    - *Révision* : changements pour l'entretien ;
    - *Transition* : changements pour s'adapter à l'environnement.

- Le deuxième niveau est constitué des « facteurs de qualité ». Les facteurs décrivent la vue des utilisateurs du logiciel et doivent être quantifiés lors de la spécification des exigences. Ils sont au nombre de 11.

- Le troisième niveau est constitué des « critères de qualité ». Les critères sont des propriétés internes qui concernent les développeurs du logiciel. Ils sont au nombre de 23.

- Le quatrième et dernier niveau concerne les métriques qui s'appliquent aux éléments mesurables et qui servent de support au contrôle de la qualité du logiciel.

Le type de relation entre un élément d'un niveau et les éléments du niveau immédiatement inférieur est une relation de contenance (c. à d. un élément d'un niveau contient les éléments du niveau suivant qui lui sont reliés).

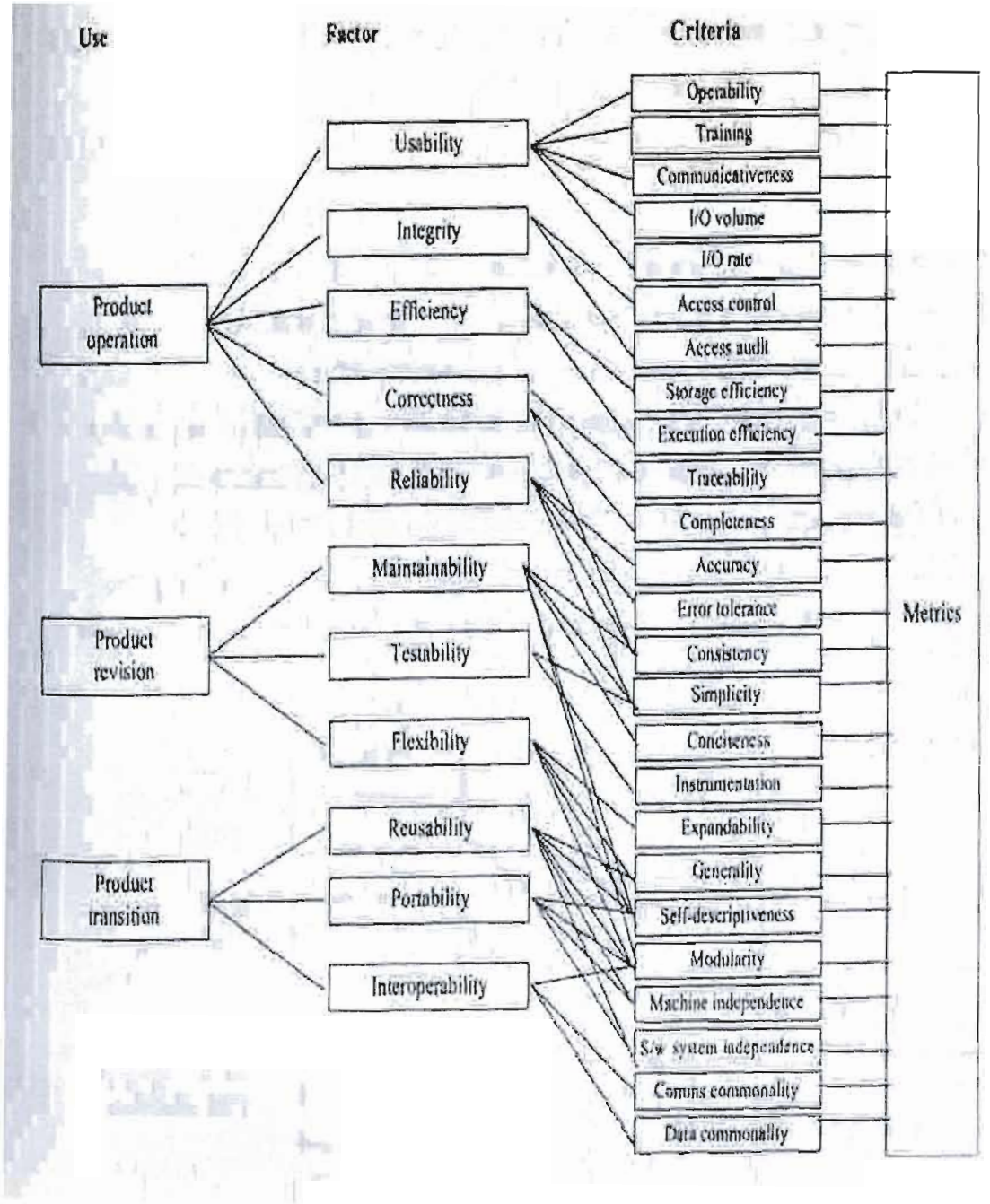


Figure 1.1 Modèle de qualité de McCall tel que présenté par Fenton (Source : Fenton et Pfleeger, 1997)

1.3 Modèle de qualité de Boehm (1978)

Pour Barry Boehm, la qualité du logiciel est quelque chose d'abstrait qui se concrétise tout au long du CVL (Boehm, 1978). Il est difficile au début de déterminer de façon exacte de quoi aura l'air le produit logiciel à développer et cela est dû aux attentes souvent différentes des parties prenantes. Selon Boehm, au tout début d'un projet, on peut se poser des questions autour de trois éléments :

- L'*utilité* du logiciel : pourquoi est-il créé ?
- La *maintenabilité*[10] du logiciel : jusqu'à quel point doit-il être facile à maintenir ?
- La *portabilité* du logiciel : doit-il rester utilisable si on change d'environnement ?

Ces trois éléments sont, pour Boehm, les trois piliers de la construction du logiciel, qui soutiennent sept facteurs de la qualité. Ces facteurs vont apparaître dans les étapes les plus avancées de la construction du logiciel et détaillent un peu plus l'utilité, la maintenabilité et la portabilité de ce dernier. Ces facteurs sont à leur tour subdivisés en des caractéristiques élémentaires aptes à être mesurées. La figure ci-dessous présente le modèle de qualité de Boehm.

[10] Même si le syntagme « facilité de maintenance » serait bien plus approprié, nous employons cet affreux terme car il fait désormais partie du jargon du GL.

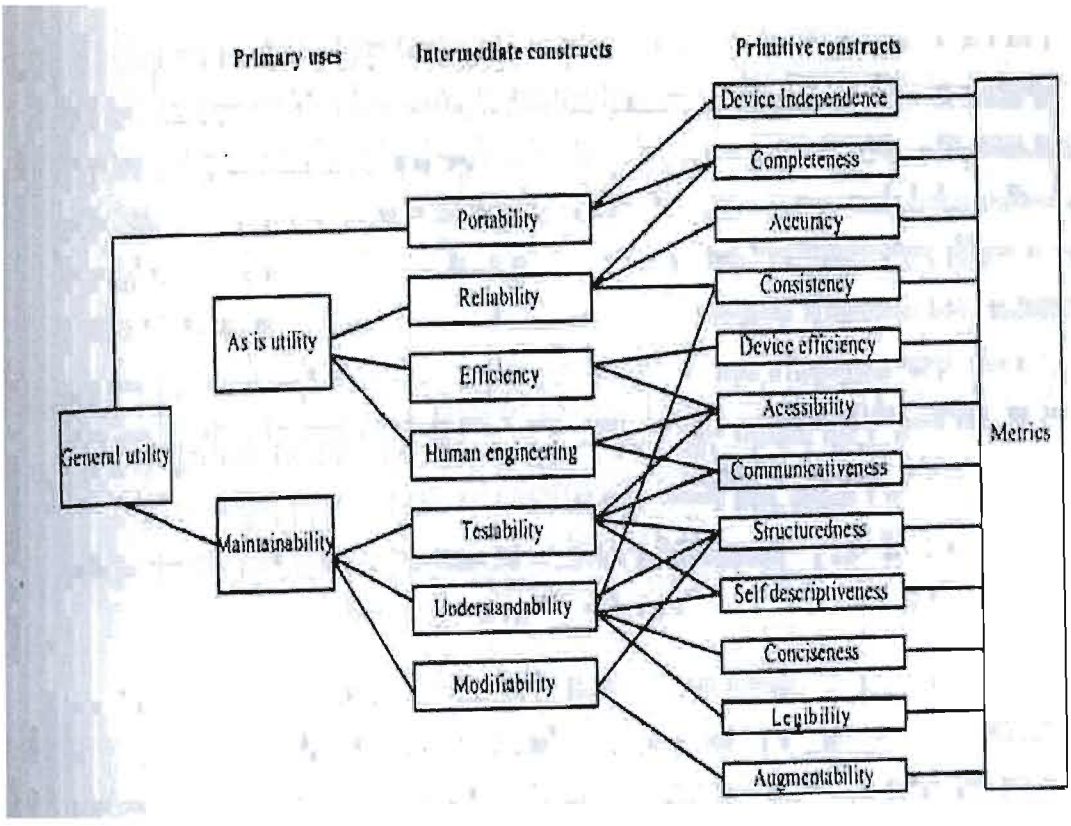


Figure 1.2 Modèle de qualité de Boehm tel que présenté par Fenton (Source : Fenton et Pfleeger, 1997)

1.4 Modèle de qualité d'ISO/IEC 9126-1

La norme ISO/IEC 9126 est constituée de quatre documents concernant la qualité du produit logiciel :

- ISO/IEC 9126-1 : présente le modèle de qualité du produit logiciel (ISO/IEC 9126-1, 2001).
- ISO/IEC 9126-2 : présente les métriques de qualité externe (ISO/IEC 9126-2, 2003)
- ISO/IEC 9126-3 : présente les métriques de qualité interne (ISO/IEC 9126-3, 2003).
- ISO/IEC 9126-4 : présente les métriques de qualité de fonctionnement (ISO/IEC 9126-4, 2004)

Dans cette partie du mémoire nous nous intéressons seulement à la partie 1, celle qui traite du modèle.

Ce modèle présente un cadre assez large pour comprendre les optiques des différents intervenants en ce qui concerne la spécification et l'évaluation de n'importe quel type de logiciel. Comme dans le cas des modèles proposés par McCall et Boehm, il s'agit d'un modèle qui structure la qualité en une hiérarchie afin de la rendre mieux compréhensible et plus facilement mesurable et, par conséquent, plus facilement contrôlable. Cette structuration se fait selon deux (2) vues :

- *Horizontale* : elle permet de mieux comprendre ce que représentent un produit logiciel et sa qualité tout au long de son cycle de vie. Voir figure ci-dessous.
- *Verticale* : elle permet d'organiser la qualité en composantes toujours moins complexes, où le dernier niveau représente des propriétés du produit pouvant être mesurées.

1.4.1 L'approche d'ISO/IEC 9126-1

Le schéma de la figure ci-dessous montre les relations de causalité entre les processus et les différents attributs de qualité du produit logiciel.

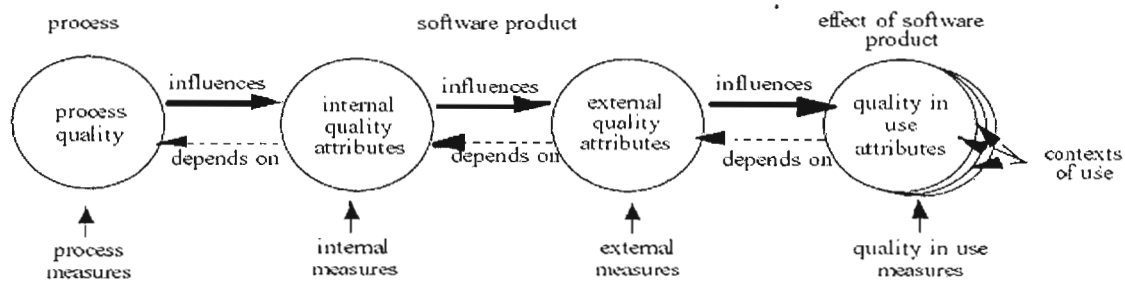


Figure 1.3 Approches de qualité[11] (Source : ISO/IEC 9126-1, 2001)

[11] Figure tirée de la section 5.1 de la première partie de la norme ISO/IEC 9126.

Comme on peut le voir dans la figure ci-dessus, la mesure de la qualité du produit logiciel (influencée par la qualité du processus) est décrite par trois (3) types d'attributs de qualité, chacun spécifié et évalué lors d'une certaine phase du CVL. Ces trois types d'attributs de qualité sont définis comme suit :

- ***Attributs de qualité interne*** : permettent de mesurer « *la qualité du produit logiciel vue de l'intérieur, sous forme de caractéristiques des produits logiciels intermédiaires, conçus avant la livraison du produit final* » (ISO/IEC 9126-1, 2001).

- ***Attributs de qualité externe*** : permettent de mesurer les caractéristiques du produit logiciel telles que vues de l'extérieur (pendant l'exécution). Ces caractéristiques sont « *évaluées et mesurées lors des tests dans un environnement similaire à celui de l'utilisateur final et avec des données simulées* » (ISO/IEC 9126-1, 2001).

- ***Attributs de qualité de fonctionnement*** : permettent de mesurer la qualité du produit logiciel final du point de vue utilisateur. « *Ce dernier est concerné surtout par l'assurance que le logiciel satisfait les buts pour lesquels il à été créé, et cela dans un environnement et un contexte d'utilisation particuliers* » (ISO/IEC 9126-1, 2001).

1.4.2 Évaluation de la qualité du produit logiciel

La définition et la structuration de la qualité sont utiles si elles favorisent l'évaluation de la qualité tout au long du cycle de vie du produit. La figure ci-dessous, tirée de la première partie de la norme, présente les liens entre les besoins (*needs*), les exigences (*requirements*) et la qualité, dans le but d'expliquer le processus d'évaluation de la qualité.

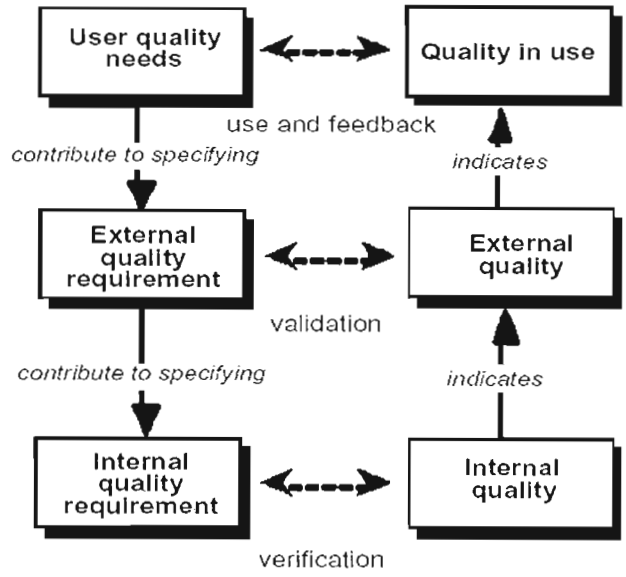


Figure 1.4 Spécification et évaluation de la qualité du logiciel[12] (Source : ISO/IEC 9126-1, 2001)

Les besoins de l'utilisateur sont la base pour l'évaluation de la qualité de fonctionnement et contribuent à la spécification des exigences de la qualité externe, qui sont utilisées comme cible pour la validation. Ces exigences sont transformées en exigences de qualité interne, qui sont utilisées dans la définition des propriétés des artefacts du logiciel et permettent aussi de fixer les critères du processus de vérification adopté.

À titre d'exemple, nous pouvons considérer un produit devant satisfaire un besoin de qualité BA. Le degré de satisfaction de ce besoin BA sera mesuré après livraison en exécutant le logiciel dans l'environnement final (*use and feedback*). En partant de BA les exigences de qualité externe EEA sont spécifiées (souvent par les analystes responsables du développement du logiciel). EEA servent à valider la qualité externe. En partant d'EEA, les exigences de qualité interne EIA sont spécifiées et la qualité interne est vérifiée par rapport à EIA.

[12] Figure tirée de la section 5.2 de la première partie de la norme ISO/IEC 9126.

C&C, les deux concepts qui sont à la base de notre étude, sont des éléments de la qualité interne. Et comme tels, ils influencent la qualité externe et donc la qualité de fonctionnement.

1.4.3 Qualité externe et interne

Même si dans la norme ISO 9126, la qualité externe et la qualité interne sont séparées, leur organisation hiérarchique et les noms des éléments qui les constituent sont les mêmes.

Le modèle de qualité externe et interne, présenté par la figure ci-dessous, organise les attributs de qualité du logiciel en six (6) catégories (caractéristiques) :

1. Capacité fonctionnelle (*functionality*)
2. Fiabilité (*reliability*)
3. Facilité d'utilisation (*usability*)
4. Rendement (*efficiency*)
5. Maintenabilité (*maintainability*)
6. Portabilité (*portability*)

Chaque caractéristique est à son tour subdivisée en un groupe de sous-caractéristiques mesurables.

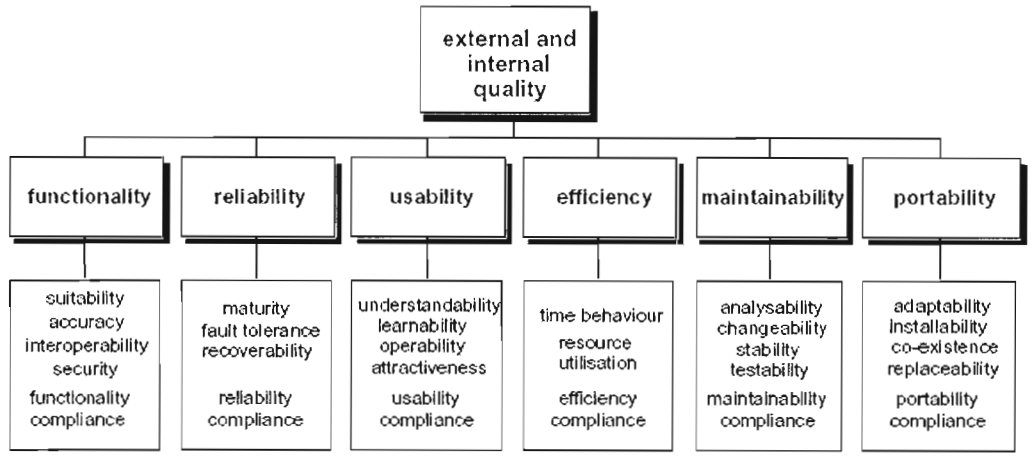


Figure 1.5 Modèle de qualité interne et externe (ISO/IEC 9126-1, 2001)

1.4.4 Qualité de fonctionnement

Le modèle de qualité de fonctionnement (voir figure ci-dessous), organise les attributs de qualité du logiciel en quatre (4) catégories (caractéristiques) :

1. Efficacité (*effectiveness*) : il s'agit de « *la capacité du produit logiciel à permettre aux utilisateurs d'atteindre les objectifs spécifiés avec exactitude et exhaustivité dans un contexte d'emploi donné* » (ISO/IEC 9126-1, 2001).

2. Productivité (*productivity*) : il s'agit de « *la capacité du produit logiciel à permettre aux utilisateurs des quantités appropriées de ressources, en relation avec l'efficacité accomplie dans un contexte d'emploi donné* » (ISO/IEC 9126-1, 2001).

3. Sûreté (*safety*)[13] : il s'agit de « *la capacité du produit logiciel à atteindre des niveaux de risque de danger pour les personnes, l'activité, le logiciel, ou l'environnement dans un contexte d'emploi donné* » (ISO/IEC 9126-1, 2001).

4. Satisfaction (*satisfaction*) : il s'agit de « *la capacité du produit logiciel à satisfaire les utilisateurs dans un contexte d'emploi donné* » (ISO/IEC 9126-1, 2001).

Contrairement aux autres types de qualité, les caractéristiques de fonctionnement ne sont pas ultérieurement organisées en sous-caractéristiques.

---

[13] À noter que dans la traduction française on tradùit « safety » avec « sécurité », ce qui est une erreur très grave surtout dans le cadre ISO où « safety » est toujours traduit par« sûreté » (et « security » par « sécurité »).

Figure 1.6 Modèle de qualité de fonctionnement[14]

1.5 Étude comparative de trois modèles

Dans cette section, nous allons comparer brièvement les trois modèles que nous venons de présenter. Après les avoir comparés du point de vue structurel, nous allons analyser plus en détail la maintenabilité.

NOTE Dans la comparaison des modèles nous ne considérons pas le premier niveau du modèle de Boehm, car il ne s'agit que d'un nom pour identifier la racine. FIN DE LA NOTE

1.5.1 Structure

Les modèles partagent une même structure hiérarchique à trois niveaux avec les métriques associées au troisième niveau.

Le tableau ci-dessous présente les syntagmes choisis par les créateurs des modèles pour nommer chaque niveau. Comme on peut le constater, les trois modèles n'ont aucun syntagme en commun. Dans les modèles de McCall et Bohem seulement, on retrouve des termes à peu près équivalents pour le premier niveau : « *Use* » et « *Primary uses* ». Le premier niveau est pratiquement le même pour le modèle de McCall et de Boehm : les deux modèles ont comme discriminateur « emploi ». La norme, par contre, au premier niveau, discrimine l'usage final (*effect of product software*) de tout ce qui concerne le développement (*product software*).

---

[14] Figure extraite de la section 7.2 de la première partie de la norme ISO/IEC 9126.

Tableau 1.1 Trois niveaux de qualité

	Niveau 1	**Niveau 2**	**Niveau 3**
McCall	*Use*	*Factor*	*Criteria*
Bohem	*Primary uses*	*Intermediate constructs*	*Primitif constructs*
ISO 9126	*Types de qualité*	*Caractéristiques*	*Sous-caractéristiques*

Ces différences terminologiques sont-elles importantes ou s'agit-il de simples idiosyncrasies langagières ? S'agit-il de synonymies parfaites ? Pour répondre à ces questions qui ressortent assez naturellement du tableau, nous allons analyser en détail la maintenabilité.

1.5.2 Comparaison de la maintenabilité

Nous avons choisi la maintenabilité parce qu'il s'agit d'un élément[15] particulièrement important pour notre mémoire : pratiquement tous les auteurs que nous avons consultés considèrent que la cohésion et le couplage influencent directement ou indirectement la maintenabilité.

1.5.2.1 Maintenabilité dans le modèle de McCall

Dans McCall, la maintenabilité est un élément du deuxième niveau de la hiérarchie (un « *factor* ») et fait partie de la « *product revision* » avec la testabilité et la flexibilité.

La figure ci-dessous, présente les composantes de la maintenabilité telles qu'extraites du modèle de McCall.

---

[15] Nous employons le terme générique « élément » pour que notre terminologie soit indépendante des modèles.

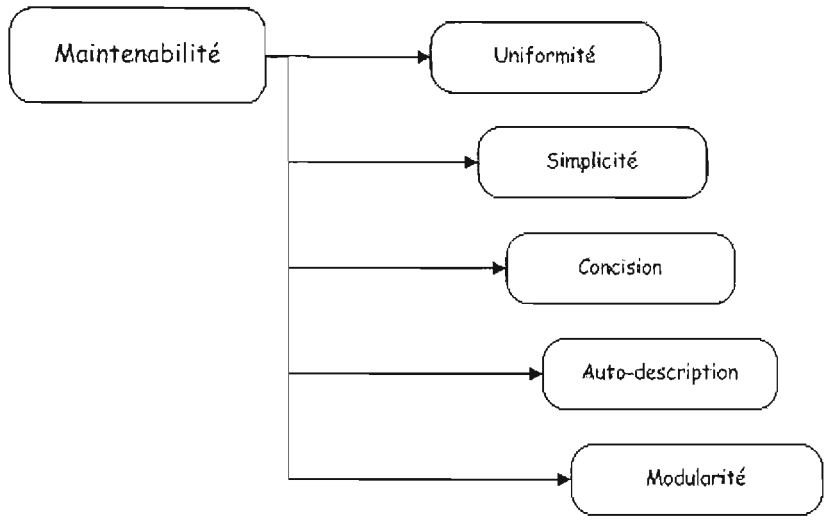


Figure 1.7 Maintenabilité dans McCall (Source : McCall, Richards et Walters, 1977)

La maintenabilité est un facteur de qualité regroupant cinq (5) critères : *Uniformité*, *Simplicité*, *Concision*, *Auto-description* et *Modularité*. La modularité est particulièrement importante pour notre étude car, comme on le verra dans le chapitre suivant, la cohésion et le couplage sont des critères qui ont un impact énorme sur la qualité de la division d'un système en modules. Si on garde constants tous les autres critères de la maintenabilité, on peut donc dire qu'en améliorant la modularité, on rend le logiciel plus facilement modifiable et donc les révisions du produit plus faciles.

1.5.2.2 Maintenabilité dans le modèle de Boehm

Dans Boehm, la maintenabilité est un élément qui se retrouve au premier niveau[16] (*Primary uses*) avec la « *As is Utility* ». La maintenabilité est constituée de trois éléments de niveau 2 (voir figure ci-dessous) :

- Facilité de compréhension du logiciel.
- Facilité de test du logiciel.

---

[16] Par-dessus la maintenabilité, à un niveau que nous pourrions appeler 0 (zéro), il y a « utilité générale ». Nous avons l'impression que ce « niveau 0 » a été introduit pour que la portabilité (qui se trouve au deuxième niveau) ne soit pas orpheline.

- Facilité de modification du logiciel.

Cela ne nous est d'aucune utilité pour comparer les deux modèles. Mais si l'on considère non pas les composantes de la maintenabilité mais les composantes de ses composantes, on retrouve trois éléments qui ont exactement les mêmes noms que ceux de McCall : *Auto-description*, *Uniformité* et *Concision*. La modularité dont nous avons souligné l'importance dans la section précédente est absente du modèle de Boehm. Effectivement, il n'existe aucun élément du troisième niveau nommé « modularité » mais il en existe un, « structuration », qui, à notre avis, pourrait être considéré comme un synonyme de « modularité »[17].

[17] Pour voir le niveau de synonymie, il faudrait considérer les métriques mais cela mériterait presque un mémoire en soi.

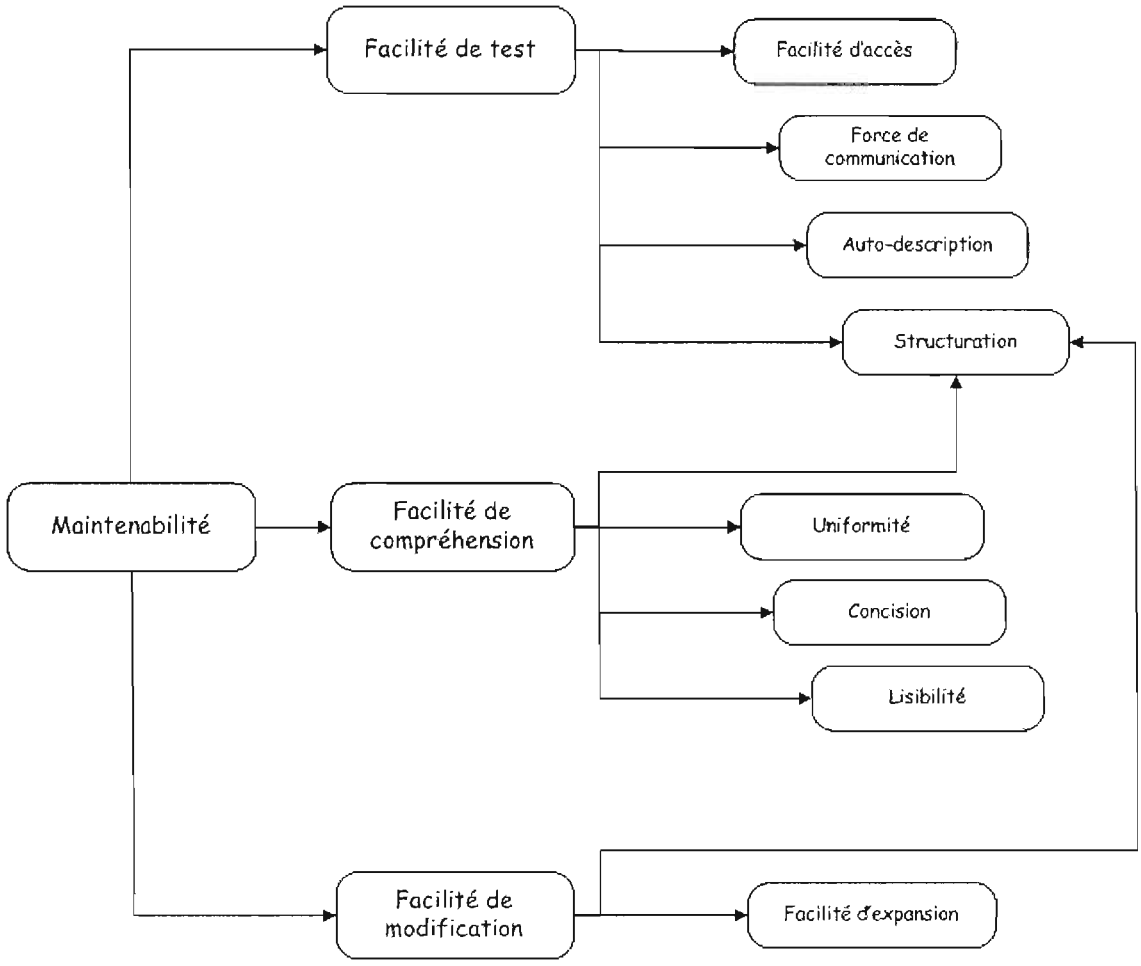


Figure 1.8 Maintenabilité dans Boehm (Source : Boehm, 1978)

1.5.2.3 Maintenabilité dans ISO/IEC 9126-1

La maintenabilité est une caractéristique de la qualité interne et externe mais n'apparaît pas dans la qualité de fonctionnement : ce qui est « naturel » car la maintenabilité concerne surtout les réalisateurs du logiciel et indirectement (par la qualité externe) le personnel qui réalise les essais.

La maintenabilité est une caractéristique (niveau 2) et se situe donc au même niveau que la maintenabilité dans le modèle de McCall. La figure ci-dessous présente les cinq (5) composantes de la maintenabilité.

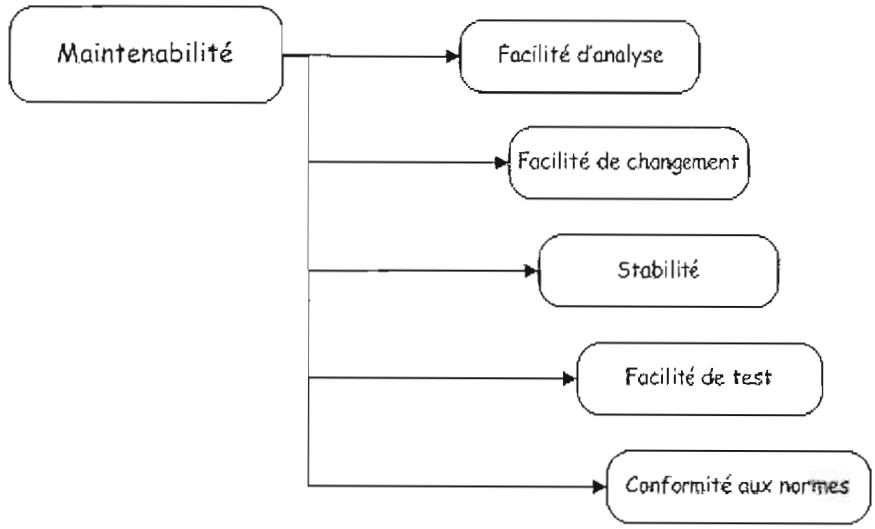


Figure 1.9 Maintenabilité dans ISO/IEC 9126-1 (Source : ISO/IEC 9126-1, 2001)

À noter que même si aucune des sous-caractéristiques[18] de la norme n'apparaît au troisième niveau des autres modèles, la facilité de test apparaît au deuxième niveau du modèle de Boehm. Parmi les sous-caractéristiques de la norme apparaît la « conformité aux normes », ce qui est fort naturel pour un modèle créé par une organisation de normalisation !

1.5.3 Conclusion

Pour faciliter la lecture de cette section nous allons employer la terminologie de la norme ISO 9126 en utilisant comme traducteur le Tableau 2.1 (p. 43).

Boehm a détaillé la maintenabilité plus que McCall et ISO/IEC 9126-1, puisqu'à cette « sur-caractéristique » correspondent indirectement huit (8) sous-caractéristiques au lieu des cinq (5) de McCall et des cinq (5) d'ISO/IEC 9126-1.

Pour Boehm ces « sous-caractéristiques » sont en effet des caractéristiques qui sont, à leur tour, divisées en sous-caractéristiques. Cela, en principe, devrait avoir comme effet un nombre plus grand de métriques propres à la maintenabilité de Boehm par rapport à la maintenabilité du modèle d'ISO/IEC 9126-1 et du modèle de McCall.

---

[18] Plus correctement il faudrait dire « aucun des noms ».

En effet, pour la maintenabilité, ISO/IEC 9126-2 propose 16 métriques et ISO/IEC 9126-3 en propose 10. Dans le livre de Boehm sur les caractéristiques de qualité du logiciel (Boehm, 1978), nous avons compté 69 métriques pour la maintenabilité. À noter que le nombre total de métriques proposées par Boehm est 151.

Si on « mesure » la qualité d'un modèle de qualité par le nombre de métriques associées à un concept, on peut affirmer que le modèle de Boehm, en ce qui concerne la maintenabilité, est meilleur que ceux de McCall et d'ISO 9126.

Nous allons terminer cette brève conclusion avec une question rhétorique qui souligne la thématique de fond de notre étude : si les éléments mesurables qui constituent la maintenabilité sont différents, peut-on dire que le concept de « maintenabilité » est le même pour les trois modèles ?

CHAPITRE II

ORIGINES DES CONCEPTS DE COHESION ET COUPLAGE AVEC UN EXCURSUS DANS LA « COMPLEXITÉ »

2.1 Introduction

Nous avons choisi le livre *Structured Design* de Yourdon et Constantine (1979) comme base pour notre analyse des concepts de C&C parce qu'il s'agit du premier livre qui a traité ces concepts d'une façon approfondie, et en même temps très pédagogique.

Nous ne considérons en détail que les chapitres suivants :

- Chapitre 5 : *Human Information Processing and Program Simplicity* .
- Chapitre 6 : *Coupling.*
- Chapitre 7 : *Cohesion.*

Avant d'analyser le couple de concepts qui est à la base de notre étude, nous considérons brièvement la complexité et la modularisation, deux concepts fondamentaux pour mieux comprendre les enjeux reliés à la cohésion et au couplage. Pour mieux analyser le concept « complexité », nous nous appuyons sur le livre de Françoise Détienne *Génie logiciel et psychologie de la programmation* (1998)

2.2 Complexité

Selon le Trésor de la langue française informatisé (TLFI, 2008), *complexité* signifie « *caractère de ce qui est composé d'éléments qui entretiennent des rapports nombreux, diversifiés, difficiles à saisir par l'esprit, et présentant souvent des aspects différents* ». La

complexité peut donc être considérée comme un état de ce qui est décomposable en plusieurs parties, liées entre elles par des liaisons pas nécessairement simples.

Dans la figure ci-dessous, en partant de la définition du TLF, nous avons essayé de montrer visuellement quelque chose de complexe

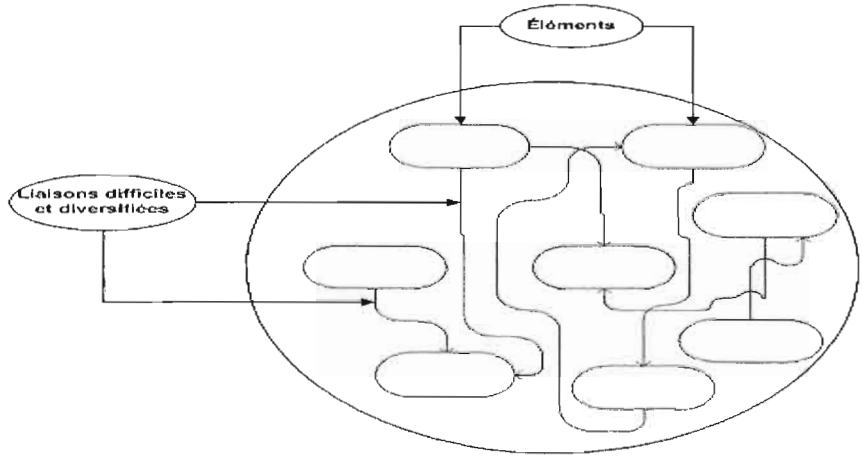


Figure 2.1 Quelque chose de complexe

En GL, il existe de nombreuses classes de complexité. À titre d'exemple en voilà trois (3) tirées du livre de Fenton et Pfleeger (1997) :

1. « *Complexité du problème* : (également appelée la complexité de calcul par des informaticiens) *mesure la complexité du problème fondamental* ».

2. « *Complexité algorithmique* : *reflète la complexité de l'algorithme mis en œuvre pour résoudre le problème ; dans un certain sens ce type de complexité mesure* ***l'efficacité*** *du logiciel* ».

3. « *Complexité structurelle* : *mesure la structure du logiciel utilisé dans la mise en œuvre de l'algorithme* ».

Cette variété n'est pas du tout étonnante si on considère :

1. le nombre élevé d'artefacts dont peut être constitué un logiciel. À titre d'exemple, la norme IEEE/EIA 12207.1 présente plusieurs dizaines de type de documents (IEEE/EIA 12207.1, 1997).

2. la complexité des processus et des activités. Qu'il suffise à ce propos de citer les soixante-quatorze activités présentées dans la norme ISO/IEC 12207 (IEEE/EIA 12207, 1995).

Peu importe la méthode employée (légère ou lourde, structurée ou axée sur les objets), le développement du logiciel passe par des « phases ». Chaque phase est caractérisé par des outils, des règles et, souvent, par des ressources humaines ayant des profils, des connaissances et des tâches différentes et qui considèrent le logiciel selon des angles différents. Ces angles, dans les meilleurs des cas, adaptent la « vision » des personnes à leurs perspectives et à leurs finalités. Par exemple, la complexité algorithmique est généralement moins importante pour l'analyste d'affaires lors de la spécification des exigences que pour le concepteur logiciel lors de la création des classes. En fonction de la phase où se retrouve le logiciel et du rôle des personnes, on peut avoir des types de complexité différents.

Yourdon et Constantine (1979) considèrent la complexité du programme qui est surtout influencée par la conception — tout leur livre traite de comment concevoir un logiciel de manière telle que les liaisons entre les parties ne soient pas trop complexes. Puisque le logiciel est constitué d'artéfacts échangés entre les personnes, ces deux auteurs mettent au centre la complexité de la perception, autrement dit, ils restreignent la complexité du logiciel à la complexité cognitive.

Par complexité cognitive, ils entendent le degré de difficulté de compréhension du logiciel, c'est-à-dire le temps et l'énergie nécessaires pour qu'un individu puisse comprendre ce que fait ce logiciel et comment il le fait. La complexité cognitive a un grand impact sur le codage et les tests, et est fortement liée à certains aspects d'apparence générale du programme (Yourdon et Constantine, 1979) :

- *La quantité d'information qui doit être bien comprise* : « *le nombre de bits de données, etc.* ».
- *L'accessibilité à cette information* : selon les auteurs « *est sans doute plus importante que la quantité d'information* ». On considère que le temps de recherche de l'information fait partie du temps nécessaire pour la comprendre. Le temps de la recherche de l'information dépend de :
    - Son mode d'accès, c'est-à-dire si la donnée est accessible à l'humain directement ou bien indirectement via d'autres données.

- Son emplacement, si on se restreint au sous-programme courant pour la trouver ou bien il faut aller chercher dans d'autres sous-programmes.
- La manière avec laquelle elle est présentée, qui peut être normalisée ou non.

- *La structure de cette information* : « *l'information est moins complexe si elle est présentée de manière linéaire, plus complexe si elle est présentée de manière imbriquée* » et elle est aussi moins complexe si on la présente dans une forme positive plutôt que négative.

Pour illustrer ces concepts, nous considérons l'exemple suivant, inspiré du calcul de la distance entre deux points, présenté aux pages 74-79 du livre que nous analysons.

Supposons que l'on nous demande d'écrire le sous-programme VITESSE, qui calcule la vitesse moyenne d'un corps dans un espace unidimensionnel. La distance parcourue par ce corps sera nommée Dis ; la durée du parcours sera nommée Dur et la formule pour le calcul de la vitesse : VITESSE = Dis / Dur.

Supposons les appels de la fonction VITESSE suivants :

1) Appel VITESSE(X, Y, Z)

Nous ne savons pas ce que représentent X, Y et Z. Un testeur peut croire que X représente la distance et Y représente la durée. Mais qui l'empêche de croire que c'est Y qui représente plutôt la durée, tandis que X représente la distance ? En ce qui concerne le troisième paramètre Z, il peut croire qu'il représente la vitesse calculée, comme il se peut qu'il soit un flag d'erreur de division par zéro. Donc, cette interface avec ses trois paramètres est susceptible d'être interprétée différemment. La complexité est due au nombre de paramètres dont le nom n'a aucun lien avec le domaine.

2) Appel VITESSE ()

Le programmeur sait très bien[19] que, pour calculer la vitesse, il a besoin de la distance et de la durée et que cette vitesse une fois calculée doit être placée quelque part. Tout cela reste invisible dans cet appel de fonction. Donc, les informations dont le programmeur a

[19] On suppose qu'il a passé son secondaire 5.

besoin sont fournies ailleurs, dans un endroit difficile à déterminer en regardant simplement cet appel de la fonction VITESSE. Ce qui rend cette interface complexe est l'accessibilité de l'information.

3) Appel VITESSE (NON-NEGATIF (CONVERSION-DISTANCE (Dis)), NON-NEGATIF (CONVERSION-DUREE(Dur)), Resultat)

Dans ce cas, la complexité est clairement due à la complexité de la structure de l'information contenue dans l'appel.

Dans la figure ci-dessous, nous avons essayé de synthétiser la complexité cognitive telle que considérée par Yourdon et Constantine (1979).

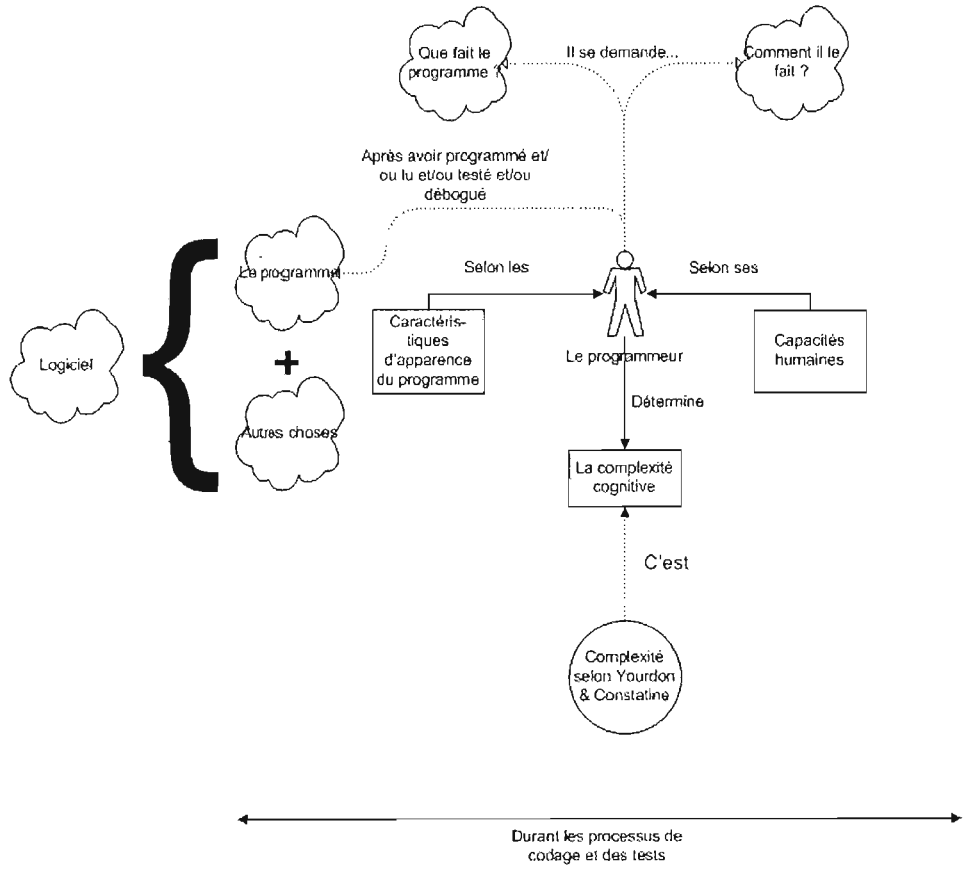


Figure 2.2 Complexité et logiciel

Lors du codage et des tests, un individu accède au logiciel sous sa forme pseudo-finale qui est le programme. Un programme est composé d'un ensemble d'instructions qui peuvent être différemment regroupées en ce qu'on appelle « modules » ou « sous-programmes ». Ces derniers représentent des « sous-fonctions », qui réalisent le but pour lequel ce logiciel a été créé. Selon la figure ci-dessus, pour qu'un programmeur puisse comprendre ce que fait un programme et comment il le fait, il doit d'abord faire appel à sa capacité de mémorisation à court terme qui est limitée[20], puis, à sa capacité de mémorisation à long terme, utilisée surtout lors de la compréhension des relations entre les différents sous-programmes. D'autres capacités comme l'attention et la vitesse de « décodage » sont sollicitées lors du processus de compréhension et affectent sa qualité (capacités humaines dans la figure ci-dessus). Les caractéristiques d'apparence du programme, (telles que la taille, les types de données, les flux de données, etc.) définissent la complexité cognitive de ce programme qui détermine la « puissance cérébrale » nécessaire pour le comprendre.

2.2.1 Complexité : approche psychologique

Dès que l'on parle de complexité cognitive, on risque de se noyer dans la psychologie. Dans le but de ne pas trop nous éloigner de notre sujet, nous avons étudié *Génie logiciel et psychologie de la programmation* de Françoise Détienne (1998). De ce livre nous n'allons considérer que les chapitres 6 et 7 qui expliquent en détails le processus de compréhension d'un programme donné.

Détienne décrit l'activité de compréhension de programme — considéré comme un texte — comme une activité qui consiste en (Détienne, 1998) :

- La construction de représentations,
- Le traitement des informations, et
- L'appel aux connaissances acquises.

---

[20] George Miller (1956) a prouvé que chez l'homme, la capacité de la mémoire à court terme, requise pour la résolution de problème manipulant plusieurs éléments à la fois, est d'environ 7 ± 2 entités (Yourdon et Constantine, 1979).

La figure ci-dessous montre comment ces trois actions permettent à un être humain de comprendre un programme. En effet, la compréhension d'un programme revient à élaborer une représentation de ce dernier, à partir des informations extraites du programme et d'autres informations extraites des connaissances de l'humain. Le fait d'évoquer les connaissances conservées en mémoire aide amplement à compléter le programme, à comprendre ou à rendre son contenu plus cohérent.

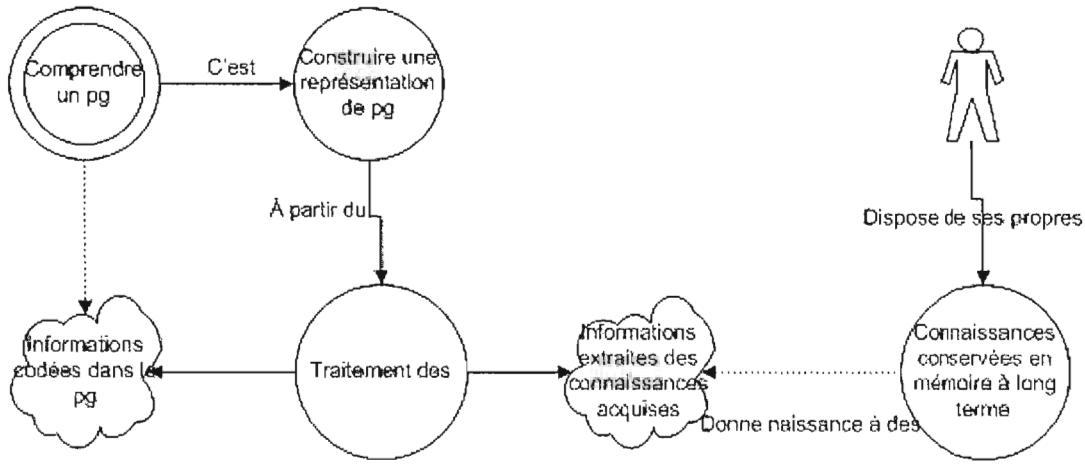


Figure 2.3 Processus de compréhension de programme

Détienne a défini quatre (4) approches théoriques pour la compréhension des programmes (Détienne, 1998) :

- ***Approche fonctionnelle*** **:** « *Selon cette approche, comprendre un programme c'est activer des schémas de connaissance, qui représentent les connaissances génériques de l'expert en informatique* » (Détienne, 1998). L'activation de ces schémas revient à affecter à leurs variables les valeurs représentées par le programme.

- ***Approche structurelle*** **:** « *La compréhension de programme correspond à la construction d'un réseau propositionnel — on considère que le texte du programme pourrait être décrit en termes d'un petit nombre de structures de contrôle. Boehm-Davis, Holt et Schultz distinguent des modes de structuration différents des programmes comme la structuration fonctionnelle et la structuration par objet* » (Détienne, 1998).

- ***Approche selon « modèle mental » :*** *« Comprendre un programme c'est construire une représentation particularisée de la situation. On distingue deux types de représentations :*

    - *Modèle du programme, similaire au concept de modèle propositionnel.*

    - *Modèle de situation, comprend des aspects statiques — qui réfèrent aux objets du problème, relations entre les objets, objets informatiques et buts principaux du programme. Et des aspects dynamiques qui représentent les communications entre les objets à un haut niveau de granularité et les relations entre les variables à un bas niveau de granularité* » (Détienne, 1998).

- ***Approche selon « résolution de problème » :*** *« Selon certains auteurs, un cadre théorique de résolution de problème rendrait mieux compte de la compréhension de programme qu'un cadre théorique de compréhension de texte. Selon une approche en termes de résolution de problème, la compréhension de programme correspondrait à des mécanismes de résolution de problème et de reconnaissance de plan. Ces auteurs soulignent l'importance des processus de sélection et du caractère sélectif des représentations construites dans la compréhension de programme*» (Détienne, 1998).

Ce qui nous semble intéressant dans ces différentes approches théoriques, c'est les phénomènes observés lors de la compréhension d'un programme selon chaque approche. Ces phénomènes se présentent sous forme de processus établis lors de chaque approche de compréhension suivie.

Comme complément à cet excursus dans le domaine de la complexité nous avons introduit l'appendice C.

2.3 Modularisation

Le terme « module » était l'un des termes les plus employés dans les premières années du GL. Un terme qui, seulement dans les dernières années, a été évincé de sa position dominante pas le terme « modèle » : *« SWEBOK, en tant que "guide pour le corpus des connaissances en génie logiciel" est un bon terrain pour vérifier les succès des mots dans la discipline. Dans les 202 pages de la version 2004, "modèle" et ses dérivés apparaissent 295 fois, tandis que "module" et ses dérivés n'apparaissent que 19 fois* » (Maffezzini, 2005).

L'une des définitions les plus générales et les plus précises de « module » est celle de Yourdon et Constantine (1979) : « *Une séquence contiguë d'énoncés d'un programme, limitée par des éléments identifiables et ayant un identifiant d'agrégat* ». Dans une approche objet, par exemple, un module peut donc être une classe, une méthode, une partie d'une méthode.

La façon de modulariser, c'est-à-dire la façon de diviser en modules un programme, a une influence directe sur les coûts de maintenance d'un logiciel. En rendant le logiciel facilement modifiable on diminue le coût de maintenance dûs aux changements des exigences du client, car on réduit l'effort humain fourni lors du développement comme lors de la validation. Et, l'effort humain est l'élément qui a le plus d'effets sur les coûts, car il affecte pratiquement tous les autres éléments (Yourdon et Constantine, 1979).

L'effort fourni par l'humain dépend de ses capacités qui sont limitées face à des problèmes très complexes. Pour rendre le problème moins complexe, depuis toujours, l'humain a recours à la méthode « *dividi et impera* » qui, en GL, devient « modularisation » ou division en petites parties. Cette division est effectuée à plusieurs reprises jusqu'à atteinte du niveau le plus bas contenant des problèmes triviaux – du point de vue de celui qui est chargé de résoudre le problème. Une fois l'ensemble des sous-problèmes résolus, leurs solutions seront placées dans des « boîtes », appelées modules, interagissant entre elles pour résoudre le problème complexe initial. Mais, cela est malheureusement trop simpliste. Nous allons suivre à la trace Yourdon et Constantine pour arriver à la figure ci-dessous qui montre comment la division en modules à partir d'un certain point peut complexifier plutôt que simplifier le problème.

Considérons le problème P et deux sous-problèmes P1 et P2 que l'on suppose prendre en charge exactement une moitié du problème : P1= P/2 et P2= P/2. Si on appelle C(P) les coûts de programmation du programme P, pour le principe « *dividi et impera* » on aura :

$$C(P) > C(P1) + C(P2)$$

Mais pour que P1 et P2 puissent résoudre le problème P, il faut ajouter leurs interactions. Ce qui, si on appelle :

- I1 : l'ensemble des interactions entre P1 et P2
- I2 : l'ensemble des interactions entre P2 et P1

Donne : C (P) > C (P1 + I1 * P2) + C (P2 + I2 * P1)

Cette inéquation est vraie si et seulement si I1 et I2 tendent vers zéro.

Donc, d'une part, en divisant un problème on le simplifie, mais de l'autre, on le complexifie à cause de l'interaction entre les parties. En augmentant le nombre de modules les interactions intra-module diminuent et tendent vers zéro, tandis que les interactions inter-modules commencent à zéro et tendent vers des valeurs élevées. Ceci est bien synthétisé dans la figure ci-dessous tirée du livre de Yourdon et Constantine (1979). Comme on peut le voir il y a un coût total minimal, là où les deux courbes se croisent.

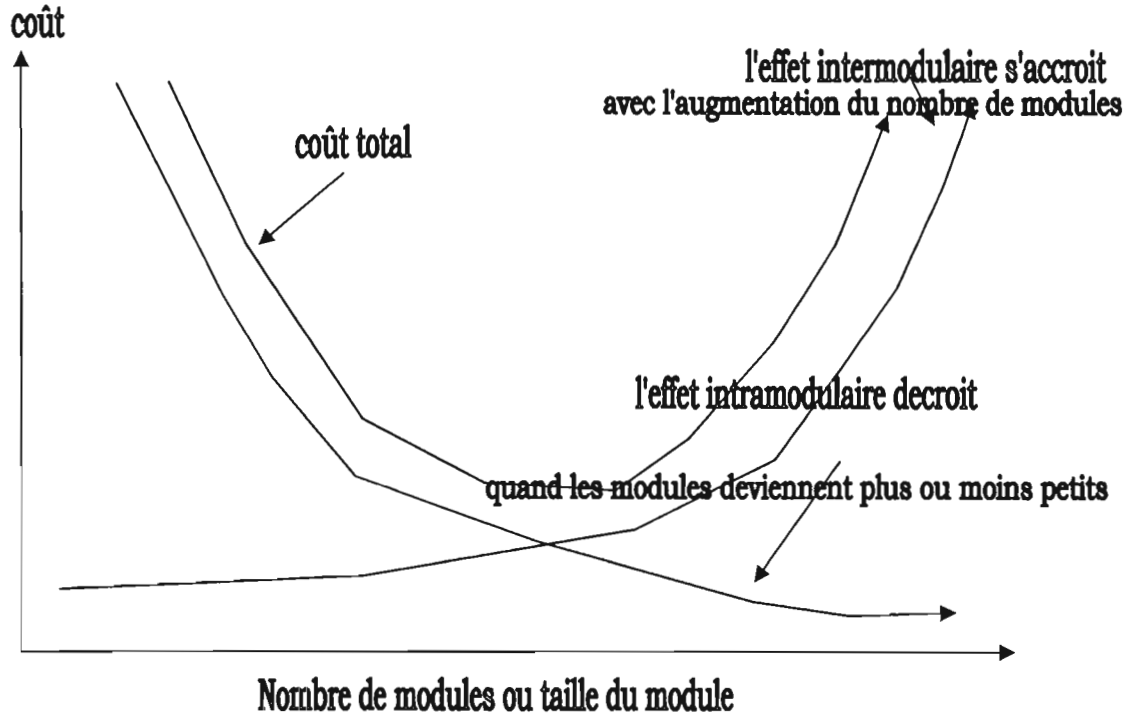


Figure 2.4 Coût du logiciel versus le nombre de modules (Source : Yourdon et Constantine, 1979)

Maintenant que nous avons introduit la modularisation et ses effets sur la complexité et sur les coûts du logiciel, nous pouvons analyser les deux concepts qui sont à la base de notre étude : le couplage et la cohésion.

2.4 Couplage

Le couplage est défini par Yourdon et Constantine (1979) comme « *le degré d'interdépendance entre les modules* ». Plus cette interdépendance est forte et plus le couplage est fort. L'interdépendance entre les modules est définie non seulement en fonction du nombre d'interconnexions, mais aussi en fonction de leur type.

Puisqu'il est évident que le nombre et le type d'interconnexions entre les modules d'un système détermine la complexité de ce dernier, on peut dire que le degré de couplage est, en quelques sorte, une mesure de la complexité du système en entier. Ce qui est intuitivement évident, c'est que le couplage a une corrélation positive avec la complexité. Nous ne craignons pas d'affirmer que s'il n'y avait pas de corrélation positive entre couplage et complexité, il faudra se questionner sur les définitions des deux termes.

Yourdon et Constantine (1979) considèrent le couplage comme une caractéristique du système définie à partir de quatre (4) sous-caractéristiques propres au même système, à savoir : le type de connexion, la complexité des interfaces, le type de flux d'information et le moment de création de la connexion. Dans ce qui suit, nous allons détailler chacune de ces quatre sous-caractéristiques.

2.4.1 Le type de connexion

Yourdon et Constantine (1979) définissent trois (3) types de connexions :

- ***Connexion minimale*** : « *Une structure est dite minimalement connectée si et seulement si elle comporte le plus petit nombre d'interconnexions et d'interfaces nécessaires pour définir le contrôle bidirectionnel et l'information transmise entre les modules communicants* ». Comme son nom le suggère, il s'agit du type de connexion qui donne un couplage minimal. Le fait que l'on ait « *le plus petit nombre* » implique qu'un couplage plus simple du point de vue du type de connexion est impossible.

- ***Connexion normale :*** Une connexion normale entre deux modules est minimale, sauf si :

  - ✓ « *Il y a plus d'un point d'entrée à un module simple, à condition que chaque entrée soit minimale en ce qui concerne les transferts de données* », ou

  - ✓ « *Le contrôle retourne à d'autres instructions au lieu de retourner aux prochains énoncés séquentiels dans le module actif, à condition que les retours alternatifs soient définis par ce module en tant qu'élément de son processus d'activation* », ou

  - ✓ « *Le contrôle est transféré à un point d'entrée normal par quelque chose autre qu'un transfert de contrôle conditionné.* » Dans ce type de connexion, on augmente le couplage car on augmente les points d'entrée et le retour n'est pas nécessairement là où on l'attend : l'instruction qui suit l'appel.

- ***Connexion anormale :*** « *Un module est dit anormalement connecté aux autres modules du système s'il n'est ni minimalement ni normalement connecté.* » Il s'agit bien sûr de la pire solution envisageable du point de vue du couplage.

2.4.2 La complexité des interfaces

Si on considère une connexion entre deux modules, le couplage est influencé par la complexité cognitive liée à la compréhension de l'interface. Comme l'écrivent Yourdon et Constantine (1979) : « *cette dimension doit tenir compte du fait qu'un appel a une procédure avec 134 paramètres est plus complexe qu'un appel à une procédure qui n'a que 2 paramètres* » (Yourdon et Constantine, 1979).

Ne compter que le nombre de paramètres n'est pas suffisant car, comme on l'a vu dans l'exemple du calcul de la vitesse, les noms des paramètres sont aussi importants. Mesurer la complexité cognitive des interfaces est loin d'être simple : « *Il y a plusieurs façon d'approximer la complexité d'une interface, mais aucune n'est parfaite* » (Yourdon et Constantine, 1979).

2.4.3 Le type d'information

Le livre présente trois (3) types de flux d'information et le degré de couplage diffère selon le type de flux d'information qui transite à travers les interfaces :

- *Donnée* : Dans le cas où seulement des données sont échangées entre les modules du système, le degré de couplage est faible. Il peut y avoir une transmission de contrôles masqués sous forme de données (souvent une variable de type booléen) qui a un effet sur le couplage, semblable à celui de la transmission de contrôle.

- *Contrôle* : Avec le flux de contrôle, la dépendance inter-modules du système s'accroît, ce qui entraîne un couplage avec un degré fort. Il s'agit du cas où un module conditionne l'évolution d'un autre module via des fanions.

- Hybride (donnée et contrôle) : Pour ce type d'information le degré de couplage devient très fort.

2.4.4 Le moment de la création de la connexion

On peut créer le lien à plusieurs moments dans le cycle de vie du logiciel. Nous présentons ci-dessous les « instants » où l'on peut créer des liens :

- Lors de l'écriture du code, en introduisant des valeurs numériques dans le code.

- Lors de la compilation, en introduisant des constantes.

- Lors de l'édition des liens entre les modules statique.

- Lors de l'exécution, avec des liens dynamiques.

- Lors de l'exécution, en établissant les liens via une base de données.

- Lors de l'exécution, via des données entrées par l'utilisateur.

Plus on s'éloigne du codage et plus le couplage diminue. Si on considère les cas extrêmes que nous venons de présenter, si le lien est dans le code, il faut recompiler (entre autre) ; s'il est créé via des données saisies par l'utilisateur, il ne faut ... rien faire.

2.5 Cohésion

Si le couplage est une caractéristique propre au système, la cohésion est une caractéristique propre à chaque module du système. Le couplage d'un module d'un système est défini à partir des connexions liant ce module aux autres modules, alors que la cohésion de ce même module est définie à partir des connexions liant les éléments internes du module.

Le couplage est une mesure de l'indépendance d'un module donné par rapport à son environnement — qui se restreint aux autres modules faisant partie du même système — et la cohésion est la mesure de l'indépendance des éléments d'un module donné. La mesure de la cohésion représente la force structurelle et fonctionnelle liant les éléments internes d'un module.

Yourdon et Constantine (1979) proposent une classification de la cohésion en sept[21] (7) niveaux :

1. ***Cohésion accidentelle :*** « *Ce niveau de cohésion correspond au point zéro de l'échelle hiérarchique de la cohésion* » (Yourdon et Constantine, 1979). La liaison entre les éléments du module est due à une modularisation aléatoire du système : on écrit du code et ensuite on l'organise en modules. C'est souvent le cas quand il n'y a aucune conception avant le codage.

2. ***Cohésion logique :*** « *Tout module dont les éléments sont logiquement groupés et qui n'exécute pas une certaine fonction est dit logiquement cohésif. Dans ce cas le module n'est qu'un « sac », où les programmeur mettent toutes les fonctions qui répondent à des exigences très génériques* » (Yourdon et Constantine, 1979). Yourdon et Constantine citent comme exemple un module pour la Gestion des fichiers.

3. ***Cohésion temporelle :*** « *On dit qu'un module est temporellement cohésif si tous ses éléments sont associés en fonction du temps* » (Yourdon et Constantine, 1979). Les procédures d'initialisation qui initialisent plusieurs éléments fonctionnellement disparates sont de bons exemples de cohésion temporelle.

4. ***Cohésion procédurale :*** « *Les éléments d'un module de cohésion procédurale forment une unité procédurale telle qu'une boucle, un énoncé de décision ou une séquence linéaire d'opérations* » (Yourdon et Constantine, 1979). Il s'agit d'une cohésion qui découle naturellement d'une approche algorithmique : c'est-à-dire d'une approche où on met au centre le flux du contrôle et non le flux des données. Pour Yourdon et Constantine, il s'agit de la dernière des cohésions négatives. À partir de la cohésion communicationnelle, on passe dans le domaine des cohésions positives.

5. ***Cohésion communicationnelle:*** « *Un ensemble d'éléments de traitement sont associés de façon communicationnelle [quand] tous les éléments opèrent sur*

[21] Il s'agit de la classification à laquelle sont confrontés pratiquement tous les chercheurs qui abordent la cohésion.

les mêmes données d'entrée et/ou génèrent les mêmes données de sorties » (Yourdon et Constantine, 1979). Il s'agit du premier niveau dans lequel « *on rencontre des relations entre les éléments de traitement qui sont intrinsèquement dépendantes du problème* » (Yourdon et Constantine, 1979). Avec cette cohésion Yourdon et Constantine donnent aussi, pour la première fois, une méthode « objective » pour faciliter l'obtention d'au moins une cohésion communicationnelle : les flux de données. Mais des modules ayant une cohésion communicationnelle peuvent avoir des coûts d'entretien très différents en fonction des types de données qu'ils reçoivent. Ce qui revient à dire que si les données sont « mal » structurées, le module sera difficile à maintenir, comme s'il était un module à cohésion plus faible.

6. ***Cohésion séquentielle :*** Pour un module ayant une cohésion séquentielle « *les données de sortie d'un élément de traitement deviennent les données d'entrée pour le prochain élément* » (Yourdon et Constantine, 1979). Les éléments sont dans une chaîne reliée par des données. Dans ce genre de cohésion, on voit encore mieux que dans la cohésion communicationnelle les effets d'une approche à la conception axée sur les flux de données. Même si ce genre de cohésion est positif, il est loin d'être idéal car un élément tout en transformant les éléments qu'il reçoit de l'élément qui le précède peut faire n'importe quoi, c'est-à-dire faire plusieurs fonctions pas nécessairement reliées.

7. ***Cohésion fonctionnelle :*** Il s'agit de la cohésion idéale où les défauts de la cohésion séquentielle ont disparu. Les auteurs, après avoir défini ce type de cohésion comme la cohésion d'un module « *où chaque élément de traitement est une partie constituante et essentielle pour l'exécution d'une fonction* » (Yourdon et Constantine, 1979), soulignent la circularité de la définition causée par la présence du terme « fonction » dans la définition. C'est pour cela qu'ils donnent une définition opérationnelle par la négative : « *La cohésion fonctionnelle est tout ce qui n'est pas séquentiel, communicationnel, procédural, temporel, logique ou accidentel* » (Yourdon et Constantine, 1979). Une approche par flux de données peut, avec une bonne probabilité, donner comme résultat des modules fonctionnellement cohésifs, si elle est accompagnée par des descriptions claires et précises de ce que font les modules et si ces descriptions prennent la forme d'une « *phrase impérative de structure simple, comprenant un verbe transitif simple et un objet spécifique* » (Yourdon et Constantine, 1979).

Même si Yourdon et Constantine écrivent que l'on peut associer une valeur numérique aux niveaux de cohésion (0 pour la cohésion accidentelle, jusqu'à 10 pour la cohésion fonctionnelle), ils mettent en évidence le danger de commencer à mesurer trop tôt : « *introduire de tel nombres actuellement (quand on a si peu d'expériences solides) pourrait introduire des éléments magiques dans tout le domaine de la programmation structurée* » (Yourdon et Constantine, 1979).

Comme on essayera de le montrer dans notre étude, la majorité des chercheurs essayeront d'introduire des mesures sans pour autant tomber dans la magie.

2.6 Conclusion

L'apport de Yourdon et Constantine au GL, en ce qui concerne la cohésion et le couplage, est fondamental et cela malgré toutes les critiques que dans les années suivantes on fera à leurs définitions et à leur approche. Il est fondamental car, malgré le manque de formalismes et de mesures, il s'agit d'une base de départ solide qui a permis à bien des chercheurs de bâtir des modèles et des méthodes qui ont permis au GL de sortir, lentement, de la « magie ». Même s'il s'agit d'une approche pragmatique loin de tout formalisme, les auteurs ne manquent pas de rigueur et surtout, comme dans la définition de la cohésion fonctionnelle ou lors de l'association de valeurs numériques aux niveaux de cohésion, ils observent leur travail avec lucidité et mesure, ce qui est une qualité essentielle de tout chercheur.

C'est aussi le côté pragmatique qui les a poussés à approfondir le concept de module et à introduire la complexité cognitive. Ce qui leur a permis de mettre au premier plan la conception et les méthodes pour obtenir des systèmes ayant une organisation en modules optimale. En partant de la cohésion et du couplage, ils introduisent des morphologies pour les systèmes, et des méthodes comme l'analyse transformationnelle, qui ont eu un énorme succès parmi les praticiens, surtout en informatique de gestion.

Nous terminons cette brève présentation en soulignant quelques éléments essentiels de l'approche des auteurs, tels qu'ils ressortent surtout des chapitres qui traitent des techniques de conception. Il nous semble important de les présenter car nous avons l'impression que, parfois, les chercheurs qui ont étudié la cohésion et le couplage ont eu moins de soucis d'ensemble.

- Sans une méthode rigoureuse d'analyse et de conception on ne peut pas espérer obtenir des systèmes ayant un couplage faible entre ses parties ;
- La morphologie d'un système est un indice fiable de la qualité de la conception ;

- L'analyse et la conception centrée sur les données permettent de concevoir des systèmes ayant des modules ayant une meilleure cohésion et un plus petit couplage[22] ;

- Évaluer la cohésion ou le couplage du code n'est utile que pour l'historique des projets : il faut pouvoir déterminer les niveaux de cohésion et de couplage bien avant le codage pour pouvoir éventuellement faire des changements structurels avant l'écriture du code.

[22] C'est un premier pas vers une approche axée sur les objets.

CHAPITRE III

COHÉSION ET COUPLAGE : ANNÉES 1980-1989

3.1 Introduction

Dans ce chapitre nous présentons les articles des années 1980 qui traitent de la cohésion et le couplage, et que nous avons synthétisés dans l'Appendice A. Pour chaque article, nous présentons le but, le contenu, ainsi que des considérations et des critiques personnelles. Ce chapitre se termine par une conclusion (section IV .6).

3.2 Emerson (1984) « Une métrique discriminante de la cohésion »

3.2.1 But

Le but de cet article est de définir une métrique objective, qui capture « les différences majeures » entre les niveaux de cohésion, métrique appelée « discriminante de cohésion de module ».

3.2.2 Contenu

Emerson (1984) détecte le degré de cohésion d'un module en fonction de la façon avec laquelle « les références de données » dans les instructions peuvent être reliées aux « flux de contrôle » entre ces instructions. La métrique proposée par Emerson permet la catégorisation suivante :

1. Un module de cohésion de type I exécute des actions contribuant toutes à l'accomplissement d'une seule fonction.
2. Un module de cohésion de type II exécute des actions produites en séquence.

3. Un module de cohésion de type III exécute une action parmi d'autres choisie selon la valeur du paramètre passé.

La métrique proposée par l'auteur est une métrique de cohésion fondée sur une propriété de la théorie des graphes. Pour appliquer cette dernière sur un module donné, on procède comme suit :

1. On construit le graphe de flux du module dont chaque sommet représente une instruction exécutable, et les arcs découlant d'un sommet relie ce dernier aux instructions qui peuvent s'exécuter immédiatement après son exécution. On ajoute un sommet terminal de façon qu'aucun arc ne sorte de ce nœud.

2. À partir de ce graphe, on construit le graphe de flux réduit, en éliminant tous les sommets (instructions exécutables) qui ne contiennent pas de référence à des variables.

3. En utilisant le graphe de flux réduit, on détermine l'ensemble des références de chaque variable du module. Un ensemble de référence d'une variable donnée est l'ensemble de sommets qui correspondent aux instructions exécutables du module qui font référence à cette variable.

4. Enfin, on calcule la moyenne de ces ensembles de référence ce qui donne la valeur de la métrique.

La formule de calcul de cette métrique avec toutes les hypothèses, les théorèmes et les preuves sont présentés dans l'Appendice A.

Puisque les niveaux de cohésion « *ne sont pas formellement définis* », aucune dérivation mathématique des limites de classe discriminantes n'est possible. Cependant, dans le tableau ci-dessous, l'auteur présente les heuristiques élaborées pour chaque niveau de cohésion :

Tableau 3.1 Heuristiques correspondant aux niveaux de cohésion d'Emerson (1984)

Type de cohésion	Valeur approximative de la métrique
Type I	À peu près (q/c)/s Avec : – q, le nombre moyen de références de variables par instruction exécutable – c, constante de proportionnalité du nombre d'instructions exécutables par rapport au nombre de variables d'un module – s, le nombre d'instructions exécutables qui référencient une variable
Type II	Lorsque le nombre d'instructions augmente, la cohésion diminue plus rapidement que 1/s
Type III	Lorsque le nombre d'instructions augmente, la cohésion diminue plus rapidement que 1/s à la puissance 2

3.2.3 Considérations et critiques

Dans cet article, on peut remarquer que le processus d'application de la métrique ne comprend aucune étape qui fasse appel à une action subjective : c'est ce qui permet à Emerson (1984) d'atteindre son but en donnant une métrique objective de la cohésion.

Le principal élément de critique de cet article est le manque de définition de la terminologie employée par l'auteur. En effet, Emerson s'est préoccupé de définir tous les éléments et toutes les notions qu'il a utilisés dans l'élaboration de sa métrique, ainsi que tout ce qui peut être nécessaire pour l'application de cette dernière, sans définir ce sur quoi il applique sa métrique « module ». Dans l'introduction, il présente indirectement le terme « module » comme une partie d'un programme. Et ce n'est que lorsqu'il a présenté sa métrique qu'il a défini ce terme.

Ce que nous reprochons aussi à Emerson, c'est le manque de précision, car nous ne savons pas quelles sont les variables concernées par la métrique de cohésion proposée. Il dit : « [...] *nous aurons besoin de définir les variables dans le module* [...] ». Mais, nous ignorons

s'il parle de toutes les variables référencées par les instructions de ce module, ou bien uniquement des variables définies dans ce module. Ce qui nous met dans cette ambigüité, c'est qu'il est courant qu'un module manipule des variables qui ne lui appartiennent pas.

Pour conclure la critique de cet article, nous pensons que l'auteur n'a pas vraiment validé la métrique qu'il a proposée. Tout ce qu'il a fait, c'est l'appliquer sur un ensemble de programmes tirés d'un livre. Comme résultat de cette application, la métrique classe ces programmes comme des modules de cohésion Type I, ce qu'Emerson estime indirectement être une preuve de validité de la métrique. Or, les modules choisis sont utilisés pour enseigner la conception logicielle, donc il est normal qu'ils soient fortement cohésifs. Cela est une façon de démontrer l'applicabilité de la métrique, mais non pas sa validité, c'est-à-dire s'assurer que la métrique fournit le bon résultat pour les autres types de cohésion.

3.3 Card, Page et McGarry (1985) « Critères de modularisation du logiciel »

3.3.1 But

Cet article présente une étude empirique sur « *l'efficacité* » de la cohésion et de la taille d'un module dans la réduction du coût de développement et du taux d'erreur du logiciel (« *ces erreurs sont les erreurs qui ont été détectées à partir de l'accomplissement des tests unitaires jusqu'à la fin des tests d'acceptation* »).

3.3.2 Contenu

Dans cette étude empirique, Card, Page et McGarry (1985) analysent l'efficacité de la cohésion et de la taille comme « critères » de la modularisation logicielle. Pour ce faire, ils analysent la cohésion et la taille de 453 modules FORTRAN de cinq (5) projets de développement de logiciel, en fonction de deux indicateurs de la modularisation qu'ils ont choisis et qui sont : le coût et le taux d'erreurs de module. Les auteurs considèrent que ces deux indicateurs sont aussi des « mesures de qualité ».

Un module est considéré comme « *la plus petite unité d'un programme compilable* ». Et la catégorisation d'un module donné, selon son niveau de cohésion, est effectuée par des listes de vérification. Cette catégorisation est effectuée en fonction du nombre de

« fonctions » contenues dans le module mesuré[23]. Elle comprend trois niveaux au lieu des sept fameux niveaux de Myers (1978)[24] :

1. Cohésion forte : le module contient une seule fonction à exécuter.
2. Cohésion moyenne : le module contient deux fonctions à exécuter.
3. Cohésion faible : le module contient trois fonctions ou plus à exécuter.

Les résultats de cette étude qui nous semblent importants et qui concernent uniquement la cohésion, sont :

« *50 % des modules à cohésion forte étudiés ne contiennent aucune erreur, tandis que seulement 18 % des modules à cohésion faible ne contiennent pas d'erreurs* ». Ce premier résultat montre l'existence d'une corrélation forte entre la cohésion et le taux d'erreurs. Plus le module est fortement cohésif plus son taux d'erreurs est faible.

« *En contrôlant la taille du module, la corrélation entre la cohésion et le coût augmente de façon significative et passe de -0.19 à -0.27. En contrôlant la cohésion du module, la corrélation entre la taille et le coût augmente, et passe de -0.31 à -0.38* ». À partir de ce résultat les auteurs déduisent que les modules de forte cohésion sont généralement petits et que les modules de grande taille (indépendamment du degré de cohésion) ont un coût de développement faible.

Les bons programmeurs, c'est-à-dire ceux qui produisent des modules avec des taux d'erreurs faibles, tendent à concevoir des modules de forte cohésion. En effet, les auteurs ont considéré « *les modules de forte cohésion comme une caractéristique des programmeurs chevronnés* [...] *En général les modules à cohésion forte ont un taux d'erreurs et un coût faibles par rapport à ceux à cohésion faible* ». On peut considérer ce résultat comme une

---

[23] Selon Card, Page et McGarry (1985), l'exécution d'une seule fonction est une condition nécessaire mais pas suffisante pour que le module soit de cohésion forte.

[24] Yourdon et Constantine (1979), comme on l'a vu dans le chapitre 3, on effectué la même catégorisation que Myers (1978).

confirmation scientifique de l'intuition des informaticiens (et des idées prônées par Yourdon et Constantine (1979)).

Les résultats de l'étude qui ont rapport à la taille peuvent être consultés dans l'Appendice A.

3.3.3 Considérations et ciritques

À notre avis cette étude est d'une importance fondamentale car elle présente d'une part l'impact de la cohésion sur le coût de développement et le taux d'erreurs d'un module, et de l'autre, la corrélation entre la cohésion et la taille du module.

Les auteurs ont évalué la cohésion en fonction de listes de vérification. Puisque les auteurs ne présentent aucun exemple de listes de vérification, il est difficile pour le lecteur de donner un jugement sur la justesse et l'exactitude des résultats de cette étude empirique.

Les auteurs soulignent que les métriques existantes ne sont pas utilisables. En effet, ils disent que ces dernières n'ont pas été employées dans leur étude en raison du « *manque de simplicité dans leurs utilisation avec leurs formats actuels* ». Donc, la métrique d'Emerson (1984) tombe dans cette catégorie.

Cette étude est arrivée à des résultats importants. Parmi eux, nous en trouvons un qui peut amplement servir dans le domaine des métriques, qui lie la cohésion au taux d'erreurs et qui dit « *qu'une bonne métrique de cohésion doit avoir une corrélation forte avec le taux d'erreurs* ». Mais, malheureusement, aucune métrique de cohésion n'a été évaluée dans ce sens dans les études subséquentes.

Nous aurions aimé que les auteurs expliquent pourquoi ils ont choisi comme *mesures de la qualité du logiciel* le coût de développement et le taux d'erreurs.

Nous terminons cette synthèse avec une observation à propos de la terminologie employée. Dans l'introduction, les auteurs présentent la « maintenabilité » comme synonyme de « modifiabilité », et qui selon eux est « un attribut de logiciel », alors que la maintenabilité

est généralement plus complexe que la modifiabilité, cette dernière n'étant que l'une des composantes de la maintenabilité (ISO/IEC 9126-1).

3.4 Selbi et Basili (1988) « Analyse de la cohésion et du couplage pour un système ayant tendance à avoir des erreurs »

3.4.1 But

Le but de cet article est de « *quantifier les rapports de couplage et de cohésion, et de les employer pour identifier la structure d'un système ayant tendance à avoir des erreurs* ». Cela est vérifié par une étude empirique qui détermine l'impact du rapport couplage/cohésion d'un logiciel sur le taux d'erreurs de ce dernier et l'effort requis pour la correction de ces erreurs.

3.4.2 Contenu

L'étude de Selbi et Basili (1988) est une étude empirique faite sur un logiciel de grande envergure d'à peu près 140 000 lignes de code, écrit en quatre (4) langages de programmation : langage de haut niveau, langage pour les administrateurs du système d'exploitation, langage de spécification de l'interface utilisateur et langage assembleur. Il comprend 77 sous-systèmes, 163 fichiers contenant au total 451 sous-programmes (programme principal, procédure ou fonction).

La méthode de collecte et d'analyse de données comprend sept (7) étapes, dont les trois (3) premières constituent le paradigme but-question-métrique (Basili et Weiss, 1984) :

1. Définir les buts de la collecte et de l'analyse des données.
2. Raffiner les buts pour déterminer la liste des questions spécifiques.
3. Établir les métriques et les catégories de données appropriées. Cet ensemble de métriques est appelé « vecteur de métriques ».

Les quatre (4) autres étapes impliquent le plan d'analyse et la collecte de données, la validation, l'analyse et l'interprétation :

4. Planifier l'étude et les méthodes d'analyse statistiques.
5. Concevoir et tester le plan de collecte des données.

6. Exécuter l'investigation en concurrence avec la collecte de données et la validation.

7. Analyser et interpréter les données en termes de but-question.

Ils définissent un vecteur de métriques de sept (7) dimensions : effort, changement sans erreur, erreurs, taille, utilisation de données, exécution, environnement (Basili et Katz, 1983). Les données de ces métriques ont été collectées de deux manières :

- Formulaires permettant de mesurer des dimensions liées aux erreurs : formulaires d'inspection, rapports de dysfonctionnement du système, etc. (voir Appendice A). Les erreurs collectées sont classées selon leur degré de sévérité, l'effort requis pour leur correction, leur type et le rapporteur de l'erreur. Plusieurs données statistiques ont été extraites et constituent une partie importante des résultats de cette recherche, même si elles ne sont pas en relation directe avec le couple C&C.

- Analyse automatique des associations de données : « *les associations de données, dont des mesures qui capturent l'interaction des données à travers les différentes parties du logiciel* ». Les auteurs citent trois (3) niveaux d'associations de données :

    i. Aassociation de données potentielle

    ii. Association de données utilisée

    iii. Association de données réelle

    L'identification du niveau ou type d'association de données pour chaque sous-système requiert l'emploi de cinq (5) outils. Ces outils sont aussi présentés dans l'Appendice A.

Les données de l'analyse des associations des données permettent de définir trois (3) mesures, dont deux (2) sont importantes pour notre étude :

- Le ratio couplage/cohésion pour un sous-programme

- Le ratio couplage/cohésion pour un sous-système

Parmi les enseignements que les auteurs tirent de leur recherche, voici ceux qui nous semblent les plus intéressants:

- Les sous-programmes avec des rapports couplage/cohésion faibles ont 8.1 fois moins d'erreurs par KLOC que les procédures avec un grand rapport couplage/cohésion et les erreurs sont 27.9 fois moins coûteuses à corriger.

- Les grands sous-systèmes avec un grand rapport couplage/cohésion ont des procédures avec 4.6 fois d'erreurs par KLOC plus que les autres catégories de sous-système.

- Le groupage de sous-programmes de rapport couplage/cohésion élevé est le groupage qui a un couplage très élevé avec les autres groupages, et une cohésion très faible entre ses sous-programmes.

La conclusion à laquelle les auteurs sont parvenus est la suivante : que ce soit pour les sous-programmes ou les sous-systèmes, plus on a un rapport couplage/cohésion faible, plus le taux d'erreurs et le coût pour corriger ces erreurs seront faibles. Ce qui implique qu'une forte cohésion et un faible couplage sont des indicateurs importants d'une bonne qualité, en ce qui concerne la présence d'erreurs et la facilité de leur correction.

3.4.3 Considérations et critiques

Les résultats de cette étude rejoignent ceux de Card, Page et McGarry (1985) en ce qui concerne la relation entre la cohésion, le taux d'erreurs et le coût de correction de ces erreurs. Card, Page et McGarry n'ont pas analysé le couplage car, d'après eux, « *la mesure du couplage exige l'étude de plus qu'un module à la fois* ».

3.5 Ott et Thuss (1989) « La relation entre les tranches et la cohésion du module »

3.5.1 But

Cet article montre comment on peut se baser sur les liens qui surgissent entre les différents éléments de traitement d'un module pour définir son degré de cohésion. La méthode de « tranchage » est employée dans l'indentification de ces liens.

3.5.2 Contenu

Un module dont on veut mesurer la cohésion est vu par les auteurs comme « *une collection d'éléments de traitement qui agissent ensemble pour calculer les sorties du module* ». La mesure du « degré de relation » entre les éléments de traitement d'un module,

nous permet de déterminer le niveau de cohésion de ce dernier. À l'aide de « graphes d'éléments de traitement », Ott et Thuss (1989) classent la cohésion en quatre (4) catégories :

1. Cohésion faible : caractérise les modules qui ont plus de deux éléments de traitement différents (aucune relation entre eux).

2. Cohésion de contrôle : caractérise les modules dont les éléments de traitement font partie de structures de contrôle communes.

3. Cohésion de données : caractérise les modules dont les éléments de traitement ont en commun des données.

4. Cohésion forte : un module de cohésion forte est un module qui comprend un seul élément de traitement qui calcule une seule sortie, ou une « chaîne » d'éléments de traitement dont chaque élément calcule une sortie employé par l'élément de traitement qui vient après, pour enfin fournir la valeur de la sortie du module.

La définition formelle de « tranche » adoptée par les auteurs est tirée de l'étude de Weiser (1981) et est basée sur les graphes de flux. Donc, la méthode de tranchage qui servait d'outil de débogage et de compréhension de programmes est utilisée ici par les auteurs comme un moyen de détermination du niveau de la cohésion.

Un profil de tranche d'une variable de sortie x du module M est l'ensemble d'instructions exécutables du module M qui font référence à cette variable. Les auteurs utilisent l'intersection des profils de tranches des variables de sorties afin d'identifier la relation entre les éléments de traitement du module, c'est-à-dire le niveau de cohésion de ce dernier. Le tableau ci-dessous présente la correspondance entre la nature de l'intersection des profils de tranches et le niveau de cohésion.

Tableau 3.2 Correspondance entre la nature de l'intersection des profils de tranches et le niveau de cohésion (Source : Ott et Thuss, 1989)

Nature de l'intersection entre les profils de tranches	Niveau de cohésion
Intersection vide	Cohésion faible
Intersection qui contient des instructions de contrôle et des définitions de variables de contrôle	Cohésion de contrôle
Intersection qui contient des définitions de données	Cohésion de données
Les tranches sont complètement contenues les unes dans les autres	Cohésion forte

Toutes les définitions qui se rapportent à la technique de tranchage sont présentées dans l'Appendice A.

Dans ce qui suit, nous présentons un des exemples de mesure de la cohésion basée sur le tranchage donné par les auteurs. Le tableau ci-dessous présente les profils de tranches des variables de sortie du module SomMoyProd. La colonne numéro de ligne énumère les instructions exécutables du module. Le profil de tranche de chaque variable de sortie est présenté par la colonne dont l'en-tête comprend le nom de cette variable, le symbole « | » dans la colonne d'une variable devant un numéro de ligne désigne que cette ligne appartient au profil de tranche de cette variable. La colonne instructions contient toutes les instructions du module.

Tableau 3.3 Profils de tranches des variables de sortie du module SomMoyProd (Source : Ott et Thuss, 1981)

Numéro de ligne	Somme	Moyenne	Produit	Instructions
				Procedure SomMoyProd (N : entier;
				Var somme : entier;
				Var moyenne : entier;
				Var Produit : entier);
				Var i : entier;
				Debut
1	|	|		Somme = 0;
2			|	Produit = 1;
3	|	|	|	Pour i = 1 à n faire
4	|	|		Somme=somme+i ;
5			!	Produit=produit*i ;
				finPour
6	!			Moyenne = somme/n;
				Fin

Le résultat de l'intersection des trois profils de tranches de ce module est la ligne numéro 3 (surlignée dans le tableau ci-dessus). Puisque l'instruction se trouvant sur cette ligne est une structure de contrôle, le module SomMoyProd a une cohésion de contrôle.

3.5.3 Considérations et critiques

Les auteurs mentionnent qu'il y a eu plusieurs essais d'objectivation de C&C que ce soit pour les rendre plus précis ou pour concevoir des métriques quantitatives. Ils ont, par exemple, utilisé le graphe de références utilisé dans la métrique d'Emerson (1984). Cela nous semble être un indice important (le premier que nous avons trouvé) de la continuité et de la progression de la recherche sur la C&C.

En ce qui concerne la technique employée par les auteurs, bien qu'elle soit efficace, elle demande beaucoup d'effort car, pour chaque variable de sortie d'un module, il faut produire son profil de tranche, qui inclut toutes les lignes de codes du module qui traitent cette variable. Comment fera-t-on dans le cas d'un module de centaines d'instructions ? Il faudra donc automatiser ce processus.

D'autre part, nous remarquons que pour déterminer le degré de cohésion d'un module, il faut passer par l'examen des types de relation entre ses éléments de traitement, c'est-à-dire, voir si l'intersection de deux profils de tranches contient des instructions de contrôle ou bien des définitions de variables de contrôles ou bien des définitions de données. Nous pensons que cela revient à examiner les flux de données entre les éléments de traitement.

3.6 Conclusion

Dans les articles consultés des années 1980, la majorité des auteurs qui ont traité du couple C&C ont repris les définitions données par Yourdon et Constantine (1979), en les considérant comme « des définitions fondamentales ». Or, C&C, telles que présentées par Yourdon et Constantine, étaient fondées sur une définition très générale de « module ».

Parmi les quatre (4) articles que nous avons traités pour cette décennie :

- Trois (3) articles considèrent la cohésion comme étant la caractéristique fondamentale pour évaluer la modularité – cohésion, qu'Emerson (1984) et Card, Page et McGarry (1985) considèrent comme un critère, et Ott et Thuss (1989) comme une propriété.

- Deux (2) articles considèrent la cohésion et le couplage comme des critères majeurs pour évaluer la qualité de la modularisation.

Rien de nouveau, comme l'écrivent les auteurs, car, bien avant, Yourdon et Constantine (1979) les ont identifiés comme les « *deux propriétés qualitatives d'une modularisation efficace* ». Donc, la cohésion et le couplage sont restés des éléments constitutifs de la qualité d'une organisation en modules d'une application. Dans la figure ci-dessous, nous avons essayé de rendre les liens entre C&C, modularité et qualité.

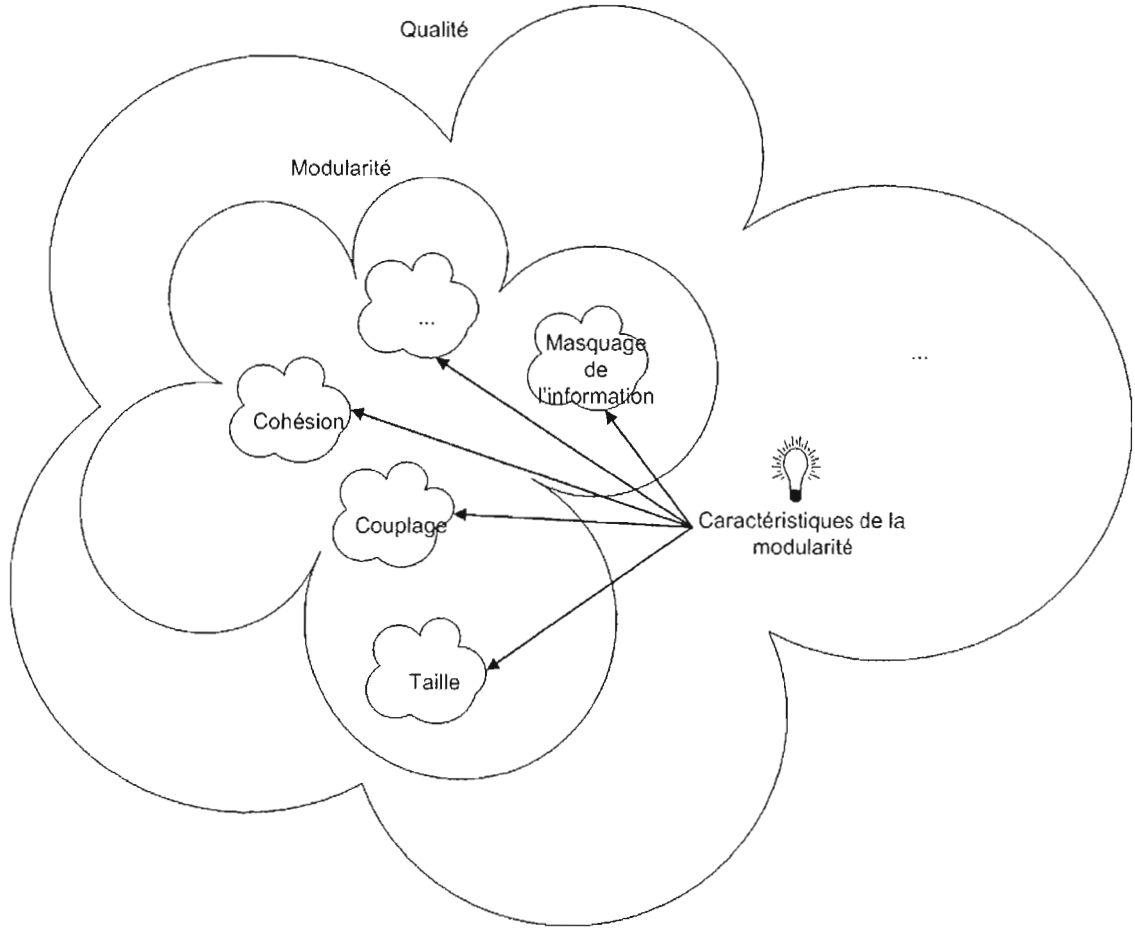


Figure 3.1 Relation entre Modularité et C&C

Tous les articles des années 1980 que nous avons analysés critiquent la façon avec laquelle Yourdon et Constantine (1979) ont traité les concepts de C&C. Les critiques sont de deux sortes :

- Critiques aux définitions : emploi de termes imprécis (pouvant avoir plusieurs définitions), comme par exemple : module, élément de traitement, liaison, etc.

- Critiques du mesurage : métriques subjectives, dans la mesure où elles requièrent l'interprétation de celui qui mesure.

- Critique des niveaux de cohésion : trop de niveaux dont les frontières ne sont pas clairement définies.

Dans le but de diminuer l'ambiguïté et la subjectivité de C&C, les études des années 1980 que nous avons prises en considération proposent diverses alternatives, à savoir :

- Définir une (des) méthode(s) de mesurage plus objective(s) (Ott et Thuss, 1989),

- Définir une (des) métrique(s) objectives(s) (Emerson, 1984),

- Réaliser une (des) étude(s) empirique(s) permettant d'analyser la corrélation entre C&C et des caractéristiques de qualité du produit final logiciel (Card, Page et McGarry, 1985 ; Selbi et Basili, 1988).

Des auteurs, comme Emerson et Ott trouvent « *qu'en pratique ce degré de discrimination* [les sept (7) niveaux de Yourdon et Constantine] *n'est pas nécessaire* » (Emerson, 1984). Emerson (1984) et Card, Page et McGarry (1985) se sont contentés d'une liste de trois niveaux et Ott et Thuss (1989) d'une liste de quatre niveaux.

Les deux articles à notre avis qui sont les plus importants parmi ceux synthétisés sont :

- L'article d'Emerson (1984), car il s'agit du seul article qui propose une métrique de cohésion objective. Nous pensons que les auteurs ont fourni beaucoup d'effort pour présenter cette métrique, et contrairement à Card, Page et McGarry (1985), nous ne voyons pas ses difficultés d'application dans de « vrais » projets.

- L'article de Selbi et Basili (1988) est aussi d'une grande importance, car il a fourni des preuves tangibles (par des chiffres) de l'impact du rapport couplage/cohésion sur deux attributs de développement de logiciel majeurs qui sont le taux d'erreurs et le coût de fixation d'erreurs.

Il est indéniable que les travaux des années 1980, en partant d'idées vagues des années 1970, comme les suivantes, ont rendu les concepts de C&C plus précis :

1. La cohésion et le couplage constituent les critères les plus importants de la modularisation.
2. Les graphes de flux de données — utilisés par Emerson (graphe de flux réduit) et par Ott et Thuss (graphe de flux d'élément de traitement); ont une relation avec le concept cohésion.

3. La mesure des associations entre les données (*data bindings*) peut être utilisée comme base de calcul de la cohésion et du couplage.

Ont rendu les concepts de C&C plus précis.

CHAPITRE IV

COHÉSION ET COUPLAGE : ANNÉES 1990-1999

4.1 Introduction

Dans ce chapitre nous présentons les articles des années 1990 qui traitent de la C&C et que nous avons synthétisés dans l'Appendice A. Pour chaque article, nous présentons le but, le contenu, ainsi que des considérations et des critiques personnelles. Ce chapitre se termine par une conclusion (section V.9

4.2 Lakhotia (1993) « Approches basées sur les règles pour le calcul de la cohésion d'un module »

4.2.1 But

Cet article a les objectifs suivants :

- Définir formellement les syntagmes « élément de traitement » et « principe d'association ».
- Définir une approche de calcul objective de la cohésion d'un module fondée sur le type d'association des éléments de traitements.

4.2.2 Contenu

Le « principe d'association » a été proposé par Constantine dans le début des années 1970. D'après l'auteur, il s'agit de la relation qui lie un ensemble d'actions exécutées par le même module et qui « *donne la cohésion entre une paire d'éléments de traitement* ». Constantine a défini une liste de cohésion à trois niveaux, chaque niveau étant propre à un type de principe d'association particulier. Cette liste a été étendue par Yourdon, Constantine,

Stevens et Myers, pour devenir la célèbre liste de cohésion de Yourdon et Constantine (1979) que nous avons présentée dans la section III.5

Par la suite, en 1974, Stevens, Myers et Constantine ont présenté un algorithme qui décrit les étapes élémentaires pour déterminer le niveau de cohésion d'un module (Stevens, Myers et Constantine, 1974). Mais, malgré cela, la cohésion est restée une notion subjective car les définitions des termes « principe d'association » et « élément de traitement » sont informelles. Donc, pour progresser vers des métriques objectives, Lakhotia formalise ces deux définitions comme suit :

- ***Élément de traitement*** : contrairement à la définition canonique d'élément de traitement (instruction), l'auteur appelle « élément de traitement » une variable de sortie d'un module. C'est-à-dire « *une variable modifiée dans le module et qui est un paramètre passé par référence ou qui est déclarée à l'extérieur du module* ». L'auteur a choisi cette définition, car la fonctionnalité du module est déterminée en termes de ses entrées/sorties[25].

- ***Principe d'association*** : correspond à deux types de relations entre les variables de sortie : dépendance de données et dépendance de contrôle. On dit qu'une variable y a une dépendance de données à l'égard de la variable x s'il y a une chaîne définition-utilisation[26] (*def-use chain*) de l'instruction n1 qui définit x à l'instruction n2 qui définit y. Et on dit que y a une dépendance de contrôle à l'égard de x à travers une instruction n3 quand cette dernière contient un prédicat qui emploie x et que n2 (qui définit y) peut être contrôlée par le succès ou l'échec du prédicat dans n3.

Ensuite, il définit les cinq (5) règles de détermination du type d'association d'une paire d'éléments de traitement. Chacune de ces règles permet de déterminer un type de cohésion parmi les suivants :

---

[25] Conscient du changement de signification introduit, dans le résumé de l'article, l'auteur souligne qu'il ne s'agit pas de « *statement* »

[26] « *C'est une liste, de chaque utilisation d'une variable, de toutes les définitions qui atteignent cette utilisation.* » (Aho, Sethi et Ullman, 1986)

- cohésion fortuite,
- cohésion logique,
- cohésion procédurale,
- cohésion communicationnelle,
- cohésion séquentielle.

La liste de ces règles est présentée dans l'Appendice A avec un exemple de module et son graphe de dépendances de variables (dont les sommets représentent les variables de sortie et les arêtes, les dépendances de donnée et de contrôle entre ces variables).

Mis à part le problème de la subjectivité du concept de cohésion, l'algorithme de calcul de la cohésion de Stevens, Myers et Constantine (1974) ne détermine pas le degré de cohésion d'un module qui a une seule ou aucune variable de sortie. Lakhotia (1993) rectifie cela dans son algorithme de calcul de cohésion :

La cohésion du module est *fonctionnelle* si le module a une seule variable de sortie, sinon

Elle est *indéfinie*, si le module ne contient aucune variable de sortie, sinon

C'est la plus petite cohésion des paires de variables de sortie du module.

Pour valider son approche de mesurage, Lakhotia (1993) compare cette dernière à celle de Stevens, Myers et Constantine (1974) en mesurant le degré de cohésion d'un ensemble de modules selon les deux approches. L'auteur remarque que les deux fournissent des résultats identiques.

Par la suite, il la compare à deux approches de mesurage de cohésion différentes : approche d'Emerson (1984) et approche d'Ott et Thuss (1989) que nous avons étudiées dans la section IV.5. Lakothia (1993) trouve que ces deux approches fournissent dans certains cas des résultats différents par rapport à ceux trouvés par Lakhotia (1993) et Stevens, Myers et Constantine (1974).

Ott et Thuss (1989) classifient la cohésion en quatre (4) niveaux : cohésion faible, cohésion de contrôle, cohésion de données et cohésion forte. Dans sa catégorisation il repose sur l'analyse des instructions contenues dans l'intersection des profils de tranches des variables de sortie; dans le cas où cette intersection est vide alors il considère le module de cohésion faible, mais le cas d'intersection vide ne veut pas toujours dire que la cohésion est faible, car elle peut être communicationnelle lorsque les instructions réfèrent aux mêmes variables d'entrée. Donc la méthode d'Ott et Thuss (1989) peut définir un module de cohésion communicationnelle comme un module de cohésion faible, et cela parce l'intersection des profils de tranches des variables de sortie de ce module est vide.

En ce qui concerne la métrique d'Emerson (1984), Lakhotia (1993) critique aussi cette métrique puisqu'elle fournit une mauvaise catégorisation de la cohésion des modules. En effet, elle identifie des modules comme étant des modules de cohésion de type II (procédurale, temporelle) ou III (logique, fortuite) alors que selon Stevens et ses collègues et selon Lakhotia ils ne le sont pas même si leurs modélisations graphiques correspondent bien à celles de modules de type II et III.

4.2.3 Considérations et critiques

Lakhotia a ajouté à l'ensemble des métriques de cohésion existantes une nouvelle métrique objective basée sur l'analyse du type d'association entre les variables de sorties d'un module, et a critiqué deux métriques qui ont été proposées à la décennie précédente en montrant clairement leurs défauts.

Nous avons aussi noté deux points faibles majeurs dans l'article de Lakhotia :

- La mesure du degré de cohésion de quelques modules selon la méthode de Stevens, Myers et Constantine (1974) a été effectuée sans mentionner l'interprétation des syntagmes génériques « élément de traitement » et « association » qui admettent plusieurs interprétations. Lakhotia (1993) a peut-être employé ses propres définitions de ces termes, mais il ne l'a pas précisé.

- La première partie de la validation de la métrique consistait en la comparaison de ses résultats avec ceux de la méthode de Stevens, Myers et Constantine (1974). Mais

comment se fait-il que l'auteur valide sa méthode par rapport à une autre qui a déjà été jugé subjective ?

Une remarque qui nous parait importante à mentionner, c'est que dans l'introduction de son article, Lakhotia déclare que « *Stevens, Myers et Constantine considèrent la cohésion comme un bon indicateur d'attributs de qualité du logiciel comme : Modifiabilité, maintenabilité, testabilité, fiabilité, etc. Seulement, cette proposition a été acceptée par la communauté du logiciel sans être expérimentalement validée* ». Pouvons-nous dire que les études empiriques de Card, Page et McGarry (1985) et Selbi et Basili (1988) effectuées dans les années 1980 valident partiellement cette proposition, puisque ces derniers ont trouvé que le coût de développement et le taux d'erreurs sont influencés par la cohésion et que nous estimons que le coût de développement et le taux d'erreurs peuvent être considérés comme des métriques d'attributs de qualité ?

4.3 Ott et Thuss (1993) « Métriques fondées sur la méthode de tranchage pour l'estimation de la cohésion »

4.3.1 But

Analyser la possibilité que des métriques fondées sur le tranchage soient adoptées comme mesures objectives de la cohésion d'un module.

4.3.2 Contenu

Dans leur ancienne recherche, Ott et Thuss (1989) avaient montré la relation existant entre les profils de tranche d'un module et le degré de cohésion de ce dernier. Et, dans sa thèse, Longworth (1985) a démontré que certaines métriques fondées sur le tranchage, conçues auparavant par Weiser (1981) comme des métriques de complexité, peuvent être employées pour mesurer la cohésion. En se fondant sur ces deux études, Ott et Thuss (1993) choisissent ces métriques comme des métriques objectives de cohésion.

Les métriques adoptées sont :

a. Les trois métriques étudiées par Longworth (1985) : couverture (*Coverage*), chevauchement (*Overlap*) et rigidité (*Tightness*).

b. Une métrique qui provient de l'article de Weiser (1981), parallélisme (*Parallelism*)

c. Deux nouvelles métriques définies par les auteurs : CouvertureMinimale (*MinCoverage)* et CouvertureMaximale (*MaxCoverage).*

Les définitions de ces métriques avec leurs formules sont présentées dans l'AppendiceA.

Toujours dans sa thèse, Longworth (1985) a rencontré un problème majeur lorsqu'il a appliqué les métriques de Weiser (1981) sur des tranches. Il a constaté que ces métriques dépendent de la taille et de l'emplacement des éléments de traitement dans un module. Deux exemples dans l'Appendice A montrent en détail les considérations de Longworth (1985).

Pour résoudre ce problème, les auteurs ont défini des « tranches de métrique » (*Metric Slices*) ne prenant en considération que les instructions qui affectent ou sont affectées par une variable de sortie. En effet, une tranche de métrique d'une variable donnée est l'union des instructions de la tranche en arrière et celles de la tranche en avant de cette variable (Ott et Thuss, 1993).

La tranche en arrière est obtenue à partir du graphe de dépendance en scrutant ce dernier et en dépistant tous les sommets qui ont un effet sur la variable en partant du sommet de cette dernière, jusqu'à l'atteinte du sommet de départ du graphe (programme principal).

La tranche en avant est calculée toujours en utilisant le graphe de dépendance, en décelant tous les sommets qui sont influencés par la variable de la tranche prenant comme départ le sommet de cette dernière, jusqu'à l'atteinte des sommets du dernier niveau du graphe.

Le tableau ci-dessous montre les valeurs que doivent avoir les six (6) métriques proposées pour catégoriser un module selon sa cohésion, s'il est de cohésion forte, faible ou acceptable.

Tableau 4.1 Les 3 niveaux de cohésion d'un module avec les valeurs des six (6) métriques d'Ott et Thuss (1989)

Cohésion	Métriques					
	Couverture	***Chevauche-ment***	***Rigidité***	***Couverture-Minimale***	***CouvertureMa-ximale***	***Parallélisme***
Forte	1	1	1	1	1	0
Faible	0	0	0	0	0	!=0
Acceptable	]0,1[	]0,1[	]0,1[	]0,1[	]0,1[	]0,1[

4.3.3 Considérations et critiques

Ott et Thuss sont partis de leur article, qui montrait la relation entre le tranchage et la cohésion, et de la thèse de Longtworth pour nous présenter cette étude qui permet d'utiliser des métriques de complexité conçues en 1981 comme des métriques de cohésion objectives. Donc il y a une sorte de continuité de ce sujet à travers les décennies, ce qui nous fait sentir que ce chemin est encourageant et peut apporter énormément à la mesure de la cohésion.

Par contre, nous ne sommes pas convaincus de la validation de ces métriques car elles sont appliquées sur des petits programmes dont le code ne nous est pas accessible.

4.4 Woodward (1993) « Les difficultés d'utilisation de la cohesion et du couplage comme indicateurs de la qualité »

4.4.1 But

Démontrer, par une étude de cas, que les échelles qualitatives de la cohésion et du couplage sont difficiles à appliquer en pratique, à cause de leur nature subjective.

4.4.2 Contenu

L'expérience de Woodward consiste en l'identification des degrés de couplage et de cohésion d'un ensemble de modules d'un programme par un groupe d'étudiants durant trois années successives. Les degrés de cohésion et de couplage à déterminer sont ceux définis par Yourdon et Constantine (1979) (voir section III.5 Cohésion)

Les 163 étudiants qui ont participé à cette expérience ont suivi un cours de génie logiciel et des conférences sur le couple C&C avec des exemples explicatifs (Woodward, 1993). Le programme utilisé, nommé *INDENT*, est codé en Fortran et compte six modules : le programme principal (*INDENT*) et cinq sous-programmes : *PUNITS*, *REFORM*, *IDENT*, *START* et *TERMIN*.

Le tableau ci-dessous présente le pourcentage des étudiants qui ont trouvé le bon degré de cohésion de chaque module.

Tableau 4.2 Pourcentage d'étudiants qui ont trouvé le bon degré de cohésion de chaque module dans l'étude de Woodward (1993)

Nom du module	Pourcentage d'étudiants trouvant la bonne cohésion
INDENT	25.15 %
PUNITS	20.86 %
REFORM	30.67 %
IDENT	28.22 %
START	44.17 %
TERMIN	29.45 %

Et le tableau ci-dessous présente le pourcentage d'étudiants qui ont réussi à bien déterminer le type de couplage entre deux modules ou plus du programme.

Tableau 4.3 Pourcentage d'étudiants qui ont réussi à bien déterminer le type de couplage entre deux modules ou plus du programme (Source : Woodward, 1993)

Modules	Pourcentage d'étudiants qui ont déterminé le bon couplage
INDENT&START&PUNITS&TERMIN	17.17 %
PUNITS&IDENT	1ère catégorie de couplage[27] : 76.00 % 2ème catégorie de couplage : 27.00 %
PUNITS&REFORM	1ère catégorie de couplage : 73.62 % 2ème catégorie de couplage : 51.53 %
START&PUNITS	40.00 %
START&REFORM	23.00 %
PUNITS&TERMIN	37.00 %
IDENT&REFORM	23.00 %

Les résultats présentés dans les deux tableaux précédents montrent clairement que « *les étudiants ont eu bien des difficultés à déterminer les types du C&C* ».

Non seulement les échelles traditionnelles du couple C&C sont de nature subjective, mais le classement se fonde sur des idées et sur une terminologie propres aux langages procéduraux des années 1970. Donc l'auteur juge nécessaire, d'une part d'objectiver et d'automatiser les catégorisations du couple C&C et cela en adoptant des métriques objectives et normalisées, et d'autre part d'adapter ces mêmes catégorisations avec les visions des nouvelles approches de conception et de n'importe quel type de langage de programmation.

Parmi les métriques et les techniques de mesure de C&C, l'auteur cite :

---

[27] Certains modules ont plus qu'un type de couplage.

- La méthode de tranchage d'Ott et Thus (1989), qui pour Woodward (1993) semble être prometteuse

- La métrique d'Emerson (1984), qui nécessite selon Woodward (1993) des études empiriques « pour vérifier les propriétés présumées des trois types de cohésion des modules »

- La métrique C du couplage de Fenton et Melton (1990) est subjective dans la mesure où c'est au mesureur d'identifier le type (ou les types) de couplage entre deux modules. De plus, une de ses variables est ambiguë.

- La métrique de couplage de Henry et Kafura (1981) qui a été développée par la suite par Shepperd (1990), mais qui n'identifie que deux catégories de couplage : le couplage contenu et le couplage commun.

Parmi les essais de restructuration des listes traditionnelles des types de cohésion et de couplage, Woodward cite les travaux de Marco et Buxton (1987) et d'Embley et Woodfield (1988). Le nouveau type de cohésion proposé par Macro et Buxton (1987) est appelé cohésion « abstraite ». Ce niveau de cohésion est associé aux modules qui utilisent les types de données abstraits, et est plus fort que la cohésion fonctionnelle. En ce qui concerne la liste de couplage, ils y ont ajouté deux nouvelles catégories :

- Couplage de « bloc » (*block*), que nous voyons lorsque « *des blocs internes d'un programme se réfèrent à des éléments définis à des niveaux plus élevés dans la structure du bloc* ».

- Couplage « importation-exportation » (*import-export*), qui se produit « *lorsqu'un module énumère les noms des éléments auxquels il accordera l'accès externe et appelle réciproquement les modules auxquels il exige l'accès* ».

Les listes de cohésion et de couplage proposées par Embley et Woodfield (1988) sont différentes des listes traditionnelles de Yourdon et Constantine (1979). Cette nouvelle catégorisation prend en considération les types de données abstraits.

4.4.3 Considérations et critiques

À notre connaissance, Woodward a été le premier qui a essayé de prouver, par une expérimentation, ce que les chercheurs disaient depuis les années 1980 à propos de l'aspect subjectif du couple C&C qui est dû à « *l'absence de métriques quantitatives normalisées* ».

Globalement, ce que nous pouvons reprocher à Woodward, c'est le choix du programme utilisé dans son expérience qui est écrit dans un langage (FORTRAN) que les étudiants participant à l'expérience ne connaissaient pas. Nous pensons que cela a dû influencer les résultats de l'expérience, même s'il dit que ces étudiants ont été munis de toutes les informations nécessaires à la réussite et à l'accomplissement de l'expérience.

Parmi les anciennes études traitant le couple C&C comme indicateurs de qualité citées par l'auteur, deux ont attiré notre attention. La première est celle de Zweben et Troy (1981) qui a démontré l'existence d'une relation entre le taux d'erreurs et le type de couplage, et la deuxième est celle de Card, Church et Agresti (1986) où on démontre le contraire. Nous pensons que les résultats de ces deux études se contredisent nettement, mais Woodward (1993) n'a pas attiré l'attention du lecteur sur cela. Il s'est contenté de mentionner ces études, et bien d'autres, seulement pour mettre le lecteur au courant de leur existence.

4.5 Chidamber et Kemerer (1994) « Suite de métriques pour la conception orientée objet »

4.5.1 But

Le but de cet article est de développer une suite de métriques orientées objet[28] (OO) validées et pouvant être utilisées pour améliorer le processus de production du logiciel.

4.5.2 Contenu

---

[28]L'approche OO s'appuie sur la modélisation du monde réel en termes de classes d'objets, alors que les anciennes approches séparent les données des procédures.

La définition de nouvelles métriques de complexité spécifiques à l'approche de conception OO s'est avérée d'une importance primordiale. Selon les auteurs, l'avis des chercheurs est unanime à cause surtout de deux faiblesses des métriques « traditionnelles » :

1. Manque de rigueur mathématique ;
2. Manque de support des concepts OO.

Les auteurs citent cinq (5) études censées éliminer les deux défauts mentionnés ci-dessus:

- Les métriques présentées pas Morris (1988) n'ont pas été validées.
- Moreau et Dominick (1989) qui ont proposé trois métriques pour les systèmes d'information graphique OO, mais qui n'ont pas fourni des définitions formelles pouvant être examinées.
- Chidamber et Kemerer (1991), Sheetz, Tegarden et Monarchi (1992) et Whitmire (1992) qui ont proposé des métriques, mais sans aucune donnée empirique.

Malheureusement, dans toutes ces tentatives, il n'y avait pas de données empiriques permettant de vérifier les métriques définies.

C'est alors que les auteurs ont défini six (6) métriques théoriquement solides, qui prennent en considération les concepts de l'approche OO et qui sont empiriquement vérifiables. Nous nous intéressons seulement à trois d'entre elles, deux mesurant le couplage : RFC et CBO et une mesurant la cohésion : LCOM. Voici comment ces deux métriques ont été définies par les auteurs :

1. ***Métrique de couplage CBO:*** appelée « Couplage entre les classes d'objet » (*Coupling between object classes* CBO). La valeur du CBO d'une classe est égale au nombre de classes avec lesquelles elle est couplée.

2. ***Métrique de couplage RFC:*** nommée « Réponse pour une classe » (*Response For a Class RFC*). C'est l'ensemble des réponses d'une classe, qu'on calcule en

additionnant le nombre de méthodes de la classe au nombre de méthodes d'autres classes qui sont appelées lors de l'instanciation de cette dernière.

3. ***Métrique de cohésion*** **:** appelée « Manque de cohésion dans les méthodes » (*Lack of Cohesion in Methods LCOM*). Elle est définie comme la différence entre le nombre des paires de méthodes d'une classe qui n'utilisent pas les mêmes variables d'instance et le nombre de méthodes de la même classe qui utilisent les mêmes variables d'instance. Cette métrique est basée sur « *la notion du degré de similitude des méthodes*[29] ».

Pour la définition de leurs métriques, les auteurs sont partis de la méthode de Booch (1991), dont la priorité est la conception des classes. Donc, ces métriques s'appliquent au niveau classe et non pas système. Cette méthode est composée de quatre (4) étapes:

1. Première étape : Identification des classes (et des objets).
2. Deuxième étape : Identification de la sémantique des classes (et des objets).
3. Troisième étape : Identification des relations entre les classes (et les objets).
4. Quatrième étape : Mise en œuvre des classes (et des objets).

D'après les auteurs, cette méthode a comme point faible « *la non capture du comportement dynamique du système* ».

Les métriques ont été validées par rapport aux propriétés de Weyuker (1988) (dont l'article est analysé dans l'Appendice A) qui donnent un cadre formel de validation des métriques de complexité logicielle. Ces propriétés ont fait l'objet de critiques de la part de plusieurs chercheurs parmi lesquels les auteurs citent Cherniavsky et Smith (1971), deux chercheurs qui pensent que « *cette liste fournit des conditions nécessaires mais pas suffisantes pour juger qu'une métrique de complexité est bonne* ». Mais, malgré cela,

---

[29] Le degré de similitude de deux méthodes est l'intersection des deux ensembles des variables d'instance utilisées par ces méthodes.

Chidamber et Kemerer (1994) jugent que ces propriétés sont utiles et ils les utilisent après avoir apporté les modifications suivantes pour les adapter à l'approche OO :

- Les deuxième, septième et huitième propriétés de Weyuker (1988) sont éliminées,
- Ils gardent les six autres en remplaçant dans chacune d'elles le terme « module » par le mot « classe ». Une analyse de l'article dans lequel il les présente se trouve dans l'Appendice A.

Le tableau ci-dessous présente les résultats de validation des métriques CBO, RFC et LCOM, telles que présentés par Chidamber et Kemerer (1994) :

Tableau 4.4 Résultat de validation des métriques CBO et LCOM[30]

Métrique	Propriété1	Propriété2	Propriété3	Propriété4	Propriété5	Propriété6
CBO	1	1	1	1	1	0
RFC	1	1	1	1	1	0
LCOM	1	1	1	0	1	0

Les auteurs remarquent que les métriques CBO et RFC vérifient toutes les propriétés, sauf la sixième. Et, c'est le cas de toutes les métriques qu'ils ont définies, puisque la complexité d'une classe n'est pas plus grande que la complexité de cette même classe subdivisée en deux classes différentes, c'est ce que dit la sixième propriété (soient P et Q deux classes, $C(P) + C(Q) < C(P + Q)$). Les auteurs ont remarqué que la gestion de la mémoire et la détection d'erreurs deviennent plus compliquées avec un nombre élevé de

[30] Le chiffre 1 signifie que la métrique vérifie la propriété correspondante et 0 veut dire le contraire.

classes. Donc, ils estiment que cette propriété est inutile dans la validation de métriques de complexité (Chidamber et Kemerer, 1994).

En ce qui concerne la métrique LCOM, les auteurs ont remarqué qu'elle ne respecte ni la quatrième ni la sixième propriété, mais vérifie toutes les autres propriétés.

La quatrième propriété dit que la complexité d'une classe ne peut être moins faible que celle des composants de cette classe, de telle sorte que pour n'importe quelles P et Q deux classes, $P \equiv Q$ & $|P| \neq |Q|$. Les auteurs démontrent que LCOM ne vérifie pas cette propriété par l'exemple suivant : Supposant une classe P qui compte trois (3) méthodes : M1, M2 et M3, M2 et M3 utilisent les mêmes variables d'instance alors que M1 non, donc LCOM(P)=2-1=1, supposant une autre classe Q qui comprend aussi trois (3) méthodes : M4, M5 et M6, et que toutes ces méthodes utilisent les mêmes variables d'instance, dont LCOM(Q)=0, en combinant ces deux classe (P et Q), si les variables d'instance de la classe Q sont les mêmes que celles utilisées pas les méthodes : M2 et M3 —de la classe P, alors LCOM (P+Q)=0, parce que le nombre des paires de méthodes utilisant les mêmes variables d'instance dépassent celui de celles qui n'utilisent pas les mêmes variables d'instance. Alors, LCOM(P+Q) < LCOM(P), donc LCOM ne vérifie pas la quatrième propriété de Weyuker (1988).

Les auteurs ont appliqué l'ensemble des métriques sur deux projets informatiques différents. L'analyse des résultats de ces deux études empiriques sont détaillés dans l'Appendice A. À partir de ces résultats, les auteurs déterminent l'étape de COO de Booch (1991) durant laquelle chaque métrique peut être appliquée. En ce qui concerne les métriques CBO et LCOM, elles peuvent être utilisées comme suit :

- Les métriques CBO et RFC capturent l'ampleur de communication entre les classes en comptant les couples inter-classes, donc elle aussi est utilisée dans la deuxième étape de COO de Booch (1991).
- La métrique LCOM mesure la cohésion d'une classe. Elle est liée à l'empaquetage des données et les méthodes dans une définition de classe, donc elle est liée à la deuxième étape de COO de Booch (1991).

4.5.3 Considération et critiques

Cet article constitue une transformation pertinente dans le monde des métriques du fait que c'est le premier article à notre connaissance qui a présenté des métriques de complexité propres à la conception OO et les a validées.

4.6 Bieman et Kang (1998) « Mesure de la cohésion au niveau de la conception »

4.6.1 But

Dans cet article, les auteurs démontrent que la cohésion d'un module donné peut être mesurée à l'aide des seules informations disponibles lors de la conception détaillée du logiciel. Pour cela, en se basant sur deux approches connues (tranchage et association) :

1. Ils développent des mesures objectives et dont le processus de mesurage peut être automatisé,
2. ils évaluent ces mesures et
3. ils montrent comment ces mesures peuvent être utilisées pour appuyer la « conception », la « maintenance » et la « restructuration ».

4.6.2 Contenu

Quand Yourdon et Constantine (1979) ont défini la cohésion, ils l'ont considérée comme un « *attribut de conception, plutôt que de code. Cet attribut peut être employé pour prévoir des propriétés telles que la facilité de débogage, la facilité d'entretien, et la facilité de modification* ». Cependant, Bieman et Kang ont constaté que toutes les métriques de cohésion existantes ne sont applicables qu'après le codage. Cela les a poussés à concevoir des métriques de cohésion applicables au niveau de la conception détaillée.

L'information fournie au niveau de cette phase consiste en « *la spécification de l'interface du module (procédure ou fonction) et des dépendances entre les composants de cette interface* ». Les auteurs utilisent « le graphe de dépendance des entrées-sorties (*IODG Input/Output Dependence Graph*) » comme mécanisme pour saisir les données brutes.

Les deux métriques conçues par les auteurs sont :

- DLC (*Design-Level Cohesion*) : basée sur l'approche d'association entre les éléments de traitement d'un module de Lakhotia (1993), et est définie comme suit : « *Le niveau de cohésion d'un module est déterminé en fonction des niveaux de relation entre les différentes paires de sorties. La relation d'une paire de sorties est la relation la plus forte parmi toutes ses relations. Le niveau de cohésion du module est le plus faible (le niveau le plus bas) de toutes ses paires de sorties. Ce qui veut dire que la cohésion du module correspond au niveau le plus faible entre toutes ses paires* ». Le Tableau A.17 de la section A.2.5 de l'Appendice A montre pour chaque niveau de cohésion le type d'association entre une paire d'éléments de traitement.

- DFC (*Design-Level Functional Cohesion*) : basée sur la méthode de tranchage utilisée par Bieman et Ott (1994) pour la définition de trois (3) métriques de cohésion fonctionnelle :

  1. ***Cohésion lâche*** (*Loose Cohesiveness LC*), nombre relatif de composants[31] non isolés. On dit qu'un composant est isolé s'il a une relation de dépendance avec seulement une sortie du module.

  2. ***Cohésion forte*** (*Tight Cohesiveness TC*), nombre relatif de composants essentiels. Ces derniers sont des composants qui ont une relation de dépendance avec toutes les sorties du module.

  3. ***Cohésion de module*** (*Module Cohesiveness MC*), la connexité moyenne des composants du module. La connexité d'un composant est le rapport du nombre de sorties avec lesquelles ce composant a une relation de dépendance et le nombre de toutes les sorties du module.

Les formules de calcul de ces trois (3) métriques sont présentées dans l'Appendice A.

Dans leur étude, Bieman et Kang (1998) ont opté pour les approches association et tranchage, car ils jugent que les métriques basées sur ces dernières sont objectives. Pour montrer leur efficacité, ils les comparent analytiquement et empiriquement à la métrique FC (*Functional Cohesion*) de Bieman et Ott (1994) et constatent, globalement, une forte corrélation entre elles. Les tableaux de comparaison sont consultables dans l'Appendice A. Suite à cette comparaison, les auteurs arrivent à quatre (4) résultats fondamentaux :

---

[31] Les composants d'un module sont l'ensemble de ses entrées et sorties.

1. La cohésion peut être objectivement définie et mesurée au niveau de la conception.

2. Les mesures de cohésion au niveau de la conception peuvent être employées pour prévoir la cohésion au niveau du codage, car les unes correspondent « étroitement » aux autres.

3. Les mesures de cohésion au niveau de la conception peuvent être employées dans la détection de modules de conception faible et qui ont besoin d'une restructuration. DFC mesure le degré de connexion entre les composants. Elles sont tout à fait complémentaires et constituent des indicateurs totalement différents.

4. « *Le modèle d'IODG fournit un outil flexible pour une caractérisation quantitative et qualitative d'une conception logicielle. L'IODG sert de base pour toutes les mesures applicables au niveau conception. Des modèles d'IODG peuvent également être des diagrammes qui fournissent une forme de visualisation de la structure fonctionnelle d'un système* ».

4.6.3 Considérations et critiques

C'est le premier article que nous étudions qui mesure la cohésion du module à la phase conception, donc dorénavant nous pouvons dire qu'il existe des métriques de cohésion prédictives qui permettent au concepteur d'améliorer la qualité de sa conception avant d'embarquer dans le codage. L'idée en elle-même est une progression vers une clarté et une vision du concept cohésion sous un angle différent. Mais nous nous demandons si les auteurs ont réalisé leur but. Dans leur mesure de cohésion, les auteurs se sont basés principalement sur le type de liaison qui existe entre les composants et les sorties du module, qui à notre avis est déterminable à partir de l'analyse des instructions exécutées par ce module, donc on doit rentrer dans le code pour voir ce qui se passe, non pas comme pensent les auteurs se baser sur une interface avec des entrées et des sorties. Nous voyons que les auteurs ont appliqué leurs métriques sur des programmes, si ces derniers constituent vraiment le produit de la phase de conception détaillée, dans ce cas nous ne voyons pas la différence entre l'information utilisée par les métriques existantes fournie par du code et celle employée par ces nouvelles métriques fournie lors de la conception détaillée, puisqu'au niveau de ces deux phases on dispense de la même information qui est les entrées, les sorties et leurs niveaux de dépendance (que ce soit de l'interface ou du corps du module). Peut-être que le format de

cette information change mais au fond elle reste la même. Donc, nous pensons qu'il aurait fallu s'intéresser à un niveau de conception plus haut pour pouvoir dire que les métriques de cohésion avancées fournissent vraiment un outil de prévision de la qualité du logiciel.

Une autre critique que nous pouvons faire à cet article concerne l'approche de programmation des modules pour lesquels les auteurs ont conçu ces métriques. Il s'agit d'une approche de programmation procédurale, alors qu'il y a bien longtemps que l'approche OO est apparue et que des métriques qui lui sont propres existent et ont été validées, modifiées à maintes reprises et adoptées. D'ailleurs, les auteurs signalent que l'une des approches sur lesquelles ils se sont basés pour définir leurs métriques (le tranchage) a été adoptée pour développer des métriques de cohésion d'une classe donnée d'un logiciel orienté OO.

4.7 Gall, Hajek et Jazayeri (1998) « Détection du couplage logique en se basant sur l'historique des versions du produit »

4.7.1 But

Présenter une approche qui analyse l'information contenue dans l'historique des versions d'un système informatique, afin d'identifier les dépendances logiques entre ses modules et évaluer ainsi sa complexité.

4.7.2 Contenu

Dans cet article, les auteurs considèrent le couplage comme une métrique de code utilisée pour la mesure de la complexité structurelle, mais, qui, dans le cas de grands systèmes avec des millions de lignes de code, est parfois difficile à appliquer. Alors, les auteurs ont introduit une nouvelle technique de détection de couplage, qu'ils ont nommée CAESAR. La détection du couplage via cette technique s'effectue par l'analyse de l'historique des versions d'un logiciel donné.

CAESAR a été appliqué sur un logiciel dont le code source compte dix (10) millions de lignes de code et plusieurs milliers de fichiers. Les auteurs ont analysés vingt (20) versions

de ce logiciel produites durant deux ans. Cette technique ne règle pas seulement le problème de mesure de couplage au niveau de grands systèmes, mais permet aussi de déceler certaines dépendances appelées « couplage logique ». CAESAR repose sur deux processus :

1. CSA (*Change Sequence Analysis*) : se charge de dépister tous les couplages logiques « potentiels », et cela en examinant les listes de changement des versions des programmes. L'analyse des similitudes dans les séquences de changement permet de détecter des couplages logiques potentiels.
2. CRA (*Change Report Analysis*) : permet de s'assurer qu'il y a un vrai lien entre les changements des programmes et qu'il ne s'agit pas d'une simple coïncidence. C'est l'analyse du rapport de changement de chaque programme qui permet cette analyse.

Le fonctionnement détaillé de ces deux processus est présenté dans l'Appendice A.

4.7.3 Considérations et ciritques

Nous pensons que la technique proposée par Gall, Hajek et Jazayeri (1998) est optimiste, car elle ne rentre pas dans les détails du code ou de la conception. Elle accède à une quantité faible de données, à savoir : un tableau qui énumère les programmes du logiciel et les versions dans lesquelles ils ont été modifiés, avec les types de changement effectués. Donc, il n y a pas un traitement de données complexe. Nous pensons que cette technique, comme ont proposé les auteurs, peut être efficace et bénéfique dans l'évaluation et l'amélioration de la complexité structurelle et de façon générale la qualité des grands projets informatiques, l'amélioration parce qu'elle détecte les modules devant être restructurés.

4.8 Allen et Khoshgoftaar (1999) « Mesure de la cohésion et du couplage : Une approche de théorie de l'information »

4.8.1 But

Cet article propose trois (3) métriques, deux du couplage et une de la cohésion d'un système modulaire, appliquées au niveau système lors de la phase de conception, à savoir :

- Couplage inter-modules

- Couplage intra-module
- Cohésion

Ces métriques sont fondées sur la théorie de l'information et sont appliquées sur n'importe quel type de graphe représentant un système modulaire, obtenu lors de la phase de conception du logiciel.

4.8.2 Contenu

En 1999, Allen et Khoshgoftaar estiment que la cohésion et le couplage demeurent des concepts ambigus et le nombre de métriques proposées auparavant pour les mesurer reste faible. Ils définissent donc, dans leur article, trois métriques de couplage et de cohésion d'un système modulaire (couplage inter-modules, couplage-intra-module et cohésion) qui se basent sur la théorie de l'information. Les deux métriques de couplage et celle de la cohésion vérifient respectivement les propriétés de définition du couplage et de la cohésion présentées par Briand, Morasca et Basili (1996). Ces derniers ont suggéré ce cadre de définition formelle afin de rendre les concepts cohésion et couplage plus clairs. Les deux tableau ci-dessous, présentent ces propriétés avec leurs explications.

Tableau 4.3 Liste des propriétés du couplage de Briand, Morasca et Basili (1996)

Propriétés du couplage d'un système modulaire
Non négatif : Le couplage d'un système modulaire est non négatif.
Valeur nulle : Le couplage d'un système modulaire est nul si son ensemble d'arêtes inter-modules (intra-module) est vide.
Monotonicité : Ajouter une arête inter-modules (intra-module) à un système modulaire ne diminue pas son couplage.
Fusionnement des modules : Si deux modules, m1 et m2, sont fusionnés pour former un nouveau module m12, qui remplace m1 et m2, alors le couplage du système modulaire avec m12 n'est pas plus grand que le couplage du système modulaire avec m1 et m2 séparés.
L'additivité de modules disjoints : Si deux modules, m1 et m2, qui n'ont aucune arête inter-modules entre leurs sommets, sont fusionnés pour former un nouveau module m12, qui remplace m1 et m2, alors le couplage du système modulaire avec m12 est égal au couplage du système modulaire avec m1 et m2.

Tableau 4.4 Liste des propriétés de la cohésion de Briand, Morasca et Basili (1996)

Propriétés de cohésion d'un système modulaire
Non négativité et Normalisation : La cohésion d'un système modulaire appartient à un intervalle spécifié [0, Max].
Valeur nulle : La cohésion d'un système modulaire est nulle si son ensemble d'arêtes internes aux modules est vide.
Monotonicité : Ajouter une arête interne d'un module à un système modulaire ne diminue pas sa cohésion.
Fusionnement des modules : Si deux modules indépendants, m1 et m2, sont fusionnés pour former un nouveau module, m12, qui remplace m1 et m2, alors la cohésion du système modulaire avec m12 n'est pas plus grande que la cohésion du système modulaire avec m1 et m2.

Les deux métriques de couplage et celle de cohésion proposées par les auteurs vérifient respectivement les propriétés de couplage et de cohésion qui viennent d'être présentées. Comme protocole de mesure les auteurs ont choisi les graphes d'appel, dont les sommets représentent les différents composants du système qui ne sont autres que des « décisions » englobant de l'information, et les arêtes représentent les différents appels entre ces composants. D'après les auteurs, n'importe quelle abstraction logicielle graphique pouvant fournir ce genre d'information peut servir de protocole de mesure.

Les auteurs reprennent de même la définition de « protocole de mesure » donnée par Kitchenham, Pfleeger et Fenton (1995). Selon ces derniers, le protocole de mesure correspond à l'« *ensemble de conditions qui assurent une mesure cohérente et pouvant être répétée* » et est valide s'il est (Kitchenham, Pfleeger et Fenton, 1995) :

1. Indépendant de celui qui mesure et de l'environnement de mesure.
2. Compatible avec l'unité de mesure désirée et le but de l'activité de mesurage.

Les auteurs adoptent la théorie de l'information comme base ou fondement de leur instrument de mesure parce qu'ils voient que « *les décisions de conception sont incorporées dans un graphe comme de l'information* ».

Kitchenham, Pfleeger et Fenton (1995) définissent un instrument de mesure comme : « *un outil de mappage d'une abstraction de logiciel à un nombre ou à une catégorie* ».

Pour mesurer le couplage inter-modules, les auteurs utilisent un graphe qui représente seulement les liens entre les différents modules du système, appelé graphe d'arêtes inter-modules. Et les mesures du couplage intra-module et de la cohésion d'un système sont basées sur un graphe qui représente seulement les liens entre les composants des mêmes modules, nommé graphe d'arêtes intra-modulaires.

Ceci permet aux auteurs de donner les définitions pour leurs trois (3) métriques :

1. ***Couplage inter-module :*** Soit un système modulaire donné, MS, et son sous-graphe d'arêtes inter-modules, S. Le couplage inter-modules de ce système est la longueur minimale de description des relations dans S.

2. ***Couplage intra-module :*** Soit un système modulaire donné, MS, et son sous-graphe d'arêtes intra-modulaires, S'. Le couplage intra-module de ce système est la longueur minimale de description des relations dans S'.

3. ***Cohésion :*** Elle est définie comme le rapport du couplage inter-modules du système et de son couplage intra-module.

Ces métriques sont détaillées avec leurs formules de calcul dans l'Appendice A. Leurs preuves de validation théoriques en tant que métriques ayant les propriétés de C&C de Briand, Morasca et Basili (1996) sont présentées par les auteurs dans l'article.

En plus de définir leurs métriques, les auteurs comparent la métrique de couplage inter-module à une simple métrique basée sur le comptage, nommée nombre d'arêtes inter-modules, qui est aussi une métrique de couplage valide selon Briand, Morasca et Basili (1996). L'exemple étudié par les auteurs montre clairement la finesse de la métrique de couplage basée sur la théorie de l'information par rapport à celle basée sur le comptage. En effet, la métrique de comptage identifie deux modules qui ont le même nombre d'arêtes inter-modules comme des modules de même couplage, alors que la métrique basée sur la théorie de l'information trouve que même s'ils ont le même nombre d'arêtes inter-modules, ils n'ont pas le même couplage.

4.8.3 Considérations et critiques

Les métriques présentées par Briand, Morasca et Basili (1996) sont applicables lors de la phase de conception de haut niveau et sur n'importe quelle abstraction du système. Donc, elles peuvent être utilisées pour des prédictions de la qualité, alors que la majorité des anciennes métriques de C&C ont été utilisées surtout pour l'évaluation de la qualité.

De plus, il s'agit du premier article, parmi ceux étudiés, dans lequel les auteurs se sont préoccupés du respect de la nature de l'attribut mesuré. Nous croyons que toutes les métriques de cohésion exposées dans les articles précédents vérifient implicitement ces

propriétés car toutes partent des définitions du couple C&C de Yourdon et Constantine (1979).

Même si les auteurs ont proposé trois métriques de couplage et de cohésion, ils ont oublié d'indiquer ce que les valeurs de ces métriques peuvent bien signifier, c'est-à-dire qualifier le couplage et la cohésion d'un système modulaire (couplage fort, faible ou moyen) suite à l'application de ces métriques. Nous avons touché cela dans le paragraphe qui précède la conclusion, lorsqu'ils ont présenté l'exemple de calcul du couplage inter-modules, nous pouvons voir dans le tableau la valeur de cette métrique pour chacun des systèmes de l'exemple (2.76, 8.00, 16.00,..., etc.) mais aucun sens n'a été attribué à ces chiffres. Nous ne voyons pas comment ces métriques peuvent être utilisées.

Quand les auteurs ont comparé la métrique de couplage basée sur la théorie de l'information à la métrique « nombre d'arêtes inter-modules » qui se base sur le comptage, ils ont remarqué une différence dans les valeurs fournies par les deux. Même lorsque deux modules ont le même nombre d'arête inter-modules ils n'ont pas le même couplage. Il faut donc se demander sur quoi ils se sont basés pour dire qu'il y a vraiment une différence dans le degré de couplage entre ces deux modules. Nous pensons qu'il aurait fallu ajouter une comparaison avec une métrique externe.

À part cela, les métriques en elles-mêmes sont difficiles à appliquer dans un cas pratique. Leurs formules de calcul sont complexes, dans la mesure où il faut passer par le calcul de plusieurs éléments nécessaires dans le calcul de l'élément essentiel (couplage ou cohésion). Et même s'il arrive que nous les adoptons pour effectuer des prédictions, il est difficile de se fier à leurs résultats sans utiliser en parallèle d'autres métriques, car elles sont dépendantes l'une de l'autre. L'erreur dans une mesure peut se répercuter sur les autres mesures.

Nous ne pouvons pas douter que les auteurs sont conscients de l'ambigüité des concepts utilisés dans le génie logiciel et du besoin persistant d'épurer ses concepts et termes. Nous pouvons remarquer cela dans le paragraphe d'introduction où ils disent : « *la*

communauté du génie logiciel discute souvent de la conception logiciel en termes de concepts vagues, tels que la complexité, le couplage, ou la cohésion». Mais cette conscience ne les empêche pas d'utiliser des termes confus, comme « sous-système » pour désigner « module » - qui pour eux est simplement « un composant du système ».

Les trois métriques de C&C que les auteurs définissent sont applicables au niveau système. Donc, avec ces métriques, on mesure la cohésion et le couplage du système.

4.9 Conclusion

Les articles que nous avons choisi d'analyser pour les années 1990 ont soulevé certains problèmes et ont essayé d'y apporter des solutions.

L'un de ces problèmes —mettre au point des métriques de C&C objectives— a été soulevé par Woodward devant la difficulté d'identification de la cohésion et du couplage en l'absence de métriques objectives et normalisées. Parmi les auteurs des années 1990 qui se sont intéressé à cela, nous trouvons :

- Lakhotia (1993), qui a proposé une métrique de cohésion, basée sur « le principe d'association des éléments de traitements », qui nous paraît plus robuste et plus fiable que les vieilles métriques.
- Allen et Khoshgoftaar (1999), qui ont proposé deux métriques de couplage et une de cohésion, qui se basent sur « la théorie de l'information » — une approche nouvelle dans le monde des métriques de C&C.
- Ott et Thuss (1993), qui ont présenté six métriques de cohésion qui étaient à l'origine des métriques de complexité structurelle.

Le problème du manque de métriques objectives continue à exister car le nombre de métriques objectives proposées dans la décennie précédente est faible, comme le disent Allen et Khoshgoftaar (1999) et certaines d'entre elles ne sont pas fiables, comme a pu le prouver Lakhotia (1993) dans le cas de la métrique de cohésion d'Emerson (1984) et la technique de mesure de cohésion d'Ott et Thuss (1989), qui se base sur la méthode de tranchage. D'autres métriques proposées ne sont pas non plus efficaces, comme la métrique de couplage de Henry

et Kafura (1981) qui ne détecte que deux (2) types de couplage parmi six (6) : « le couplage commun » et « le couplage contenu ».

Le deuxième problème qui a été soulevé, dans cette décennie, est la non existence de métriques de C&C prédictives. Ce genre de métriques est censé employer l'information fournie lors de la phase conception et servir à l'évaluation de la complexité du logiciel avant la production du code, afin de guider les efforts de restructuration (Gall, Hajek et Jazayeri, 1998). Bieman et Kang se sont chargés de remédier à ce problème en proposant deux métriques de cohésion : une qui est similaire à celle de Lakhotia (1993) en se basant sur « le principe d'association des éléments de traitements », et une autre qui se base sur la méthode de tranchage comme la métrique de cohésion fonctionnelle de Bieman et Ott (Bieman et Ott, 1994).

Un troisième problème a été soulevé et résolu : l'incapacité des métriques existantes de détecter un couplage, autre que le couplage syntaxique. Une dépendance non syntaxique est difficile à déceler, surtout lorsqu'il s'agit d'une grande application dont le code source atteint des millions de lignes de code. Gall, Hajek et Jazayeri (1998) ont obvié à ce problème par la mise en œuvre d'une technique de détection de couplage appelée CAESAR, qui utilise l'historique des rapports de changement des programmes pour toutes les versions du logiciel et qui peut être appliquée aussi bien pendant la phase conception que celle du codage.

Le quatrième problème est né avec l'apparition de l'approche OO, puisqu'il a fallu concevoir des métriques de C&C propres à cette approche. Chidamber et Kemerer (1994) ont été les premiers auteurs à s'être intéressés à la conception de métriques de ce genre.

Nous attirons l'attention du lecteur sur le fait que les mesures de cohésion au niveau « classe » pour un logiciel OO ont bien été définies par certains auteurs (Bieman et Kang, 1995) (Mehra, 1997) (Ott, Bieman et Kang, 1995), mais malheureusement aucun des travaux de ces auteurs ne nous a été accessible.

Parmi les autres problèmes qui ont été soulevé lors de cette décennie, Woodward (1993) expose celui de la catégorisation du C&C, présentée par Yourdon et Constantine (1979) sous forme d'échelles ordinales. Cette dernière ne convient pas à n'importe quelle approche de conception ni à n'importe quel langage de programmation, puisque les personnes qui l'ont établie ont été influencées par l'approche procédurale et le langage FORTRAN, habituellement utilisés à l'époque.

Seulement, il semble que ce problème date des années 1980, car les solutions mentionnées par Woodward (1993) ont été proposées dans les années 1980. Mais les articles que nous avons choisis d'analyser pour les années 1980 n'ont pas traité ce sujet. Parmi les solutions proposées, les plus intéressantes selon Woodward (1993) sont :

- Proposer une nouvelle catégorisation qui n'a rien à voir avec l'ancienne, comme ont fait Embley et WoodField (1988).
- Garder l'ancienne catégorisation et rajouter quelques types de cohésion et autres de couplage, comme ont fait Macro et Buxton (1987).

Dans cette décennie, nous avons aussi remarqué une disparité dans les avis concernant la méthode de tranchage et la quantité et l'existence de métriques de cohésion objectives. Dans son article Lakhotia (1993) a critiqué la métrique basée sur la méthode de tranchage et a montré ses faiblesses. Dans la même année, Woodward (1993) pense que c'est une métrique prometteuse. Allen et Khoshgoftaar (1999) estiment que les tentatives de conception de métriques de cohésion objectives est faible, et citent comme exemple pour ces tentatives celle de Chidamber et Kemerer (1994). Donc, certains chercheurs croient qu'ils ont résolu le problème, alors que d'autres viennent après eux pour dire que le problème existe toujours.

Hormis les problèmes soulevés, les deux articles qui, à notre avis, sont les plus importants parmi ceux synthétisés pour cette décennie sont :

- L'article de Chidamber et Kemerer (1994), car c'est le premier travail dans lequel on propose des métriques de C&C spécifiques à l'approche de conception OO, suite aux critiques adressées aux métriques de C&C et aux catégorisations de C&C courantes en raison de leur manque de rigueur lorsqu'il s'agit d'une application OO. Une telle

étude était d'une grande utilité puisqu'il fallait, tôt ou tard, définir des métriques qui prennent en considération les mécanismes du développement OO.

- L'article de Gall, Hajek et Jazayeri (1998), dans lequel ils ont concrétisé leur idée de détection de couplage dans de grandes applications, sans rentrer dans les détails du code ou des artefacts complexes et lourds, et cela en vérifiant les rapports de changements fournis pour chaque version des programmes. La technique proposée nous paraît efficace et avantageuse pour les firmes qui s'intéressent au développement de grandes applications.

CHAPITRE V

COHÉSION ET COUPLAGE : ANNÉES 2000-2008

5.1 Introduction

Dans ce chapitre nous présentons les articles des années 2000-2008 qui traitent de la C&C et que nous avons synthétisés dans l'Appendice A. Pour chaque article, nous présentons le but, le contenu, ainsi que des considérations et des critiques personnelles. Ce chapitre se termine par une conclusion (section VI.9).

5.2 Chae, Kwon et Bae (2000) « Mesure de la cohésion pour les classes orienté-objet »

5.2.1 But

Cet article a pour objectifs de :

- Proposer une nouvelle métrique qui incorpore certaines caractéristiques des classes négligées par les métriques de cohésion existantes.
- Développer un outil de mesurage de la cohésion pour les programmes C++, nommé HYSS.
- Faire une étude de cas pour prouver l'efficacité de la métrique proposée.

5.2.2 Contenu

Les auteurs estiment que les métriques de cohésion spécifiques à la COO existantes, comme LCOM1 (*Lack of Cohesion in Methods*), LCOM2, LCOM3, LCOM4, LCOM5, Co,

Coh, LCC (*Loose Class Cohesions*) et TCC[32] (*Tight Class Cohesion*) sont faibles car elles considèrent toutes les méthodes de la même façon. Selon les auteurs, certains types de méthodes, qu'ils nomment « méthodes spéciales », ne devraient pas être considérées dans le calcul de C&C. Voici le trois (3) types de « méthodes spéciales » :

1. *Méthode d'accès* : C'est un type de méthode dont le comportement se résume à la recherche ou la mise à jour d'une variable d'instance. En conséquence, elle référencie cette variable. Habituellement ce type de méthode a l'une des deux formes suivantes :

    - Instruction d'affectation : variable-d-instance=valeur
    - Instruction de retour : return variable-d-instance

2. *Méthode de délégation* : Une méthode de délégation dans une classe réalise sa tâche en transmettant un message à un autre objet, particulièrement à une variable d'instance dans la classe. Généralement, elle réalise une sorte d'interaction avec une variable d'instance et elle a la forme suivante :

    Appel d'une méthode pour récupérer une variable d'instance : variable-d-instance [.|->] une-méthode ()

3. *Constructeur* et *Destructeur* : Un constructeur et un destructeur d'une classe doivent accéder aux variables d'instance essentielles qui requièrent une initialisation et une libération de leurs instances, faites respectivement par le constructeur et le destructeur de la classe. Les constructeurs et les destructeurs suivent une syntaxe spécifique, donc sont facilement repérables.

Ces trois types de méthodes pouvant se trouver dans n'importe quelle classe font parfois diminuer ou augmenter sa cohésion, alors qu'elles n'ont aucune relation ni avec « l'état » ni avec « le comportement » des objets de cette classe.

En outre, les métriques de cohésion citées précédemment « *ne considèrent pas les modèles ou types d'interaction entre les membres d'une classe (variables d'instance et méthodes)* », car leur critère de mesurage repose sur, soit le comptage du nombre de variables

[32] Toutes ces méthodes sont définies dans la section A.3.1.1.

d'instance référencées par des méthodes, soit le nombre de paires de méthodes partageant des variables d'instance.

Nous avons présenté dans l'Appendice A le comportement des trois types de méthodes considérés par les auteurs et l'impact de chacune d'entre elles sur les métriques de cohésion citées auparavant.

Donc pour résoudre le problème de considération des méthodes spéciales et le problème de mauvais critère de mesurage, les auteurs proposent une nouvelle métrique de cohésion qu'ils nomment CBMC (*Cohesion Based on Member Connectivity*) et qui, selon eux, résout ces problèmes. Le processus de calcul de cette métrique ignore les méthodes spéciales et a comme critère « la connectivité entre les membres d'une classe ».

La CBMC d'une classe C, notée CBMC(C), est définie comme le produit du facteur de connectivité, noté Fc(Gr(C)), et du facteur de structure, noté Fs(Gr(C)), de son graphe de référence Gr(C). Pour l'appliquer on procède comme suit :

a. Construction du graphe de référence

On construit le graphe de référence Gr(C) de la classe C sur laquelle on veut appliquer la métrique. Ce graphe représente les interactions entre les membres de la classe C.

b. Calcul du facteur de connectivité

Le facteur de connectivité du graphe de référence Fc(Gr(C)) est le rapport entre le nombre de méthodes adhésives (*glue methods*) Mg(Gr) et le nombre de méthodes normales Mn(Gr). Les méthodes normales sont les méthodes non spéciales d'une classe, et les méthodes adhésives est un sous-ensemble de méthodes qui, lorsqu'elles sont éliminées du graphe de référence, le divisent en sous-graphes de référence qu'on appelle « composants ».

Prenons comme exemple la classe *Stack* qui comporte :

- Quatre (4) méthodes spéciales : *Stack*(), ~*Stack*(), *IsEmpty*() et *GetTop*().

- Deux méthodes qui sont à la fois normales et adhésives : *Pop*() et *Push*(*int* n).

Son facteur de connectivité est égal à : 2/2 = 1 et son graphe de référence est illustré par la figure ci-dessous.

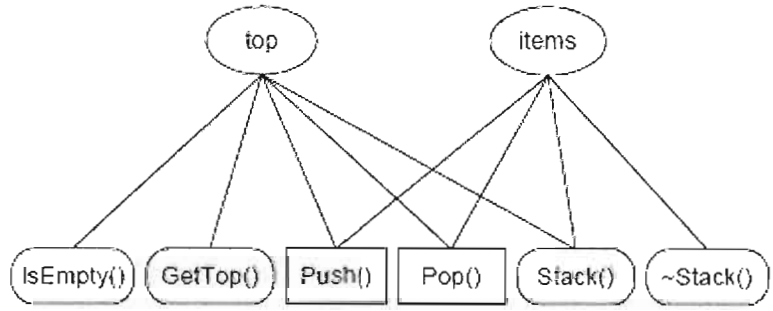


Figure 5.1 Graphe de référence de la classe *Stack* (Source : Chae, Kwon et Bae, 2000)

Les auteurs estiment que le facteur de connectivité d'une classe indique la force du lien entre ses membres.

c. Construction de l'arbre de structure de la classe

L'arbre de structure de la classe est construit à partir des composants obtenus après l'élimination des méthodes adhésives et les interactions associées du graphe de référence. La forme de l'arbre de structure d'une classe dépend de l'ordre d'élimination des méthodes adhésives. Pour cela, les auteurs suggèrent de commencer toujours par enlever la méthode adhésive dont les enfants auront une forte cohésion.

Par exemple, la figure ci-dessous illustre un exemple d'arbre de structure du graphe de référence Gr(E) d'une classe E, et qui constitue, comme on peut le voir, la racine de l'arbre. En éliminant la méthode M3 et ses interactions du graphe Gr(E), ce dernier devient divisible

en deux composants Gr1 et Gr2. Par la suite, en éliminant la méthode M2 et ses interactions, Gr1 est divisible en trois composants Gr11, Gr12 et Gr13. Par contre, le composant Gr2 n'est plus divisible car il constitue un graphe de référence MCC (*Most Cohesive Component*), puisque toutes les méthodes (normales) de ce graphe ont des interactions avec toutes ses variables d'instance, et c'est de même pour les composants Gr11, Gr12 et Gr13.

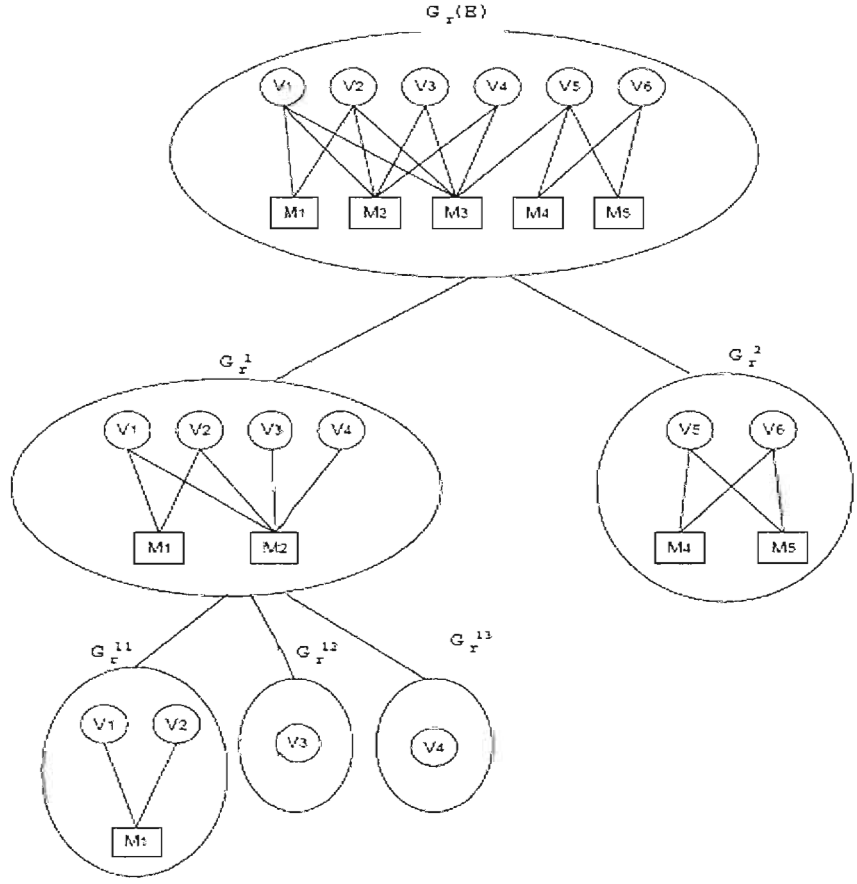


Figure 5.2 L'arbre de structure de la classe E (Source : Chae, Kwon et Bae, 2000)

d. Calcul du facteur de structure

Le facteur de structure du graphe de référence d'une classe est la cohésion moyenne des composants de ce graphe. Il dénote le degré de contribution des composants à la cohésion globale de la classe.

La qualité de conception d'une classe peut être décelée à partir des valeurs de ses facteurs de connectivité et de structure. Les auteurs identifient quatre (4) catégories de classes selon que leurs facteurs de connectivité et de structure soient bas ou élevés. Ces catégories ont été détaillées dans l'Appendice A :

1. Classe à cohésion forte (Fc et Fs élevés)
2. Classe à restructurer (Fc bas et Fs élevé)
3. Classe à cohésion faible (Fc élevé et Fs bas)
4. Classe de conception hasardeuse (Fc et Fs bas).

Dans la deuxième partie de leur article, les auteurs ont développé un outil, nommé *HYSS*, qui automatise la métrique CBMC et toutes les autres métriques de cohésion connues. Cet outil reçoit comme entrée un programme C++ et exécute un algorithme, présenté dans l'Appendice A, dont les étapes sont celles du calcul de la métrique CBMC telles que précédemment définies. Cet outil incorpore une fonction nommée *KindOfMethod* qui, pour une méthode d'une classe donnée, classifie cette dernière selon son type (normale ou spéciale).

Il est à noter que l'outil *HYSS* ne considère pas les méthodes non publiques. Après avoir présenté l'outil d'automatisation de calcul des métriques, les auteurs l'appliquent sur un système nommé *interViews*. Les résultats montrent qu'à peu prés 43 % des méthodes des classes étudiées sont spéciales. Ce qui veut dire que les méthodes spéciales sont largement utilisées et influencent donc la cohésion des classes. L'application de la métrique CBMC montre que la majorité des classes du système nécessitent une restructuration (Fc bas et Fs élevé).

Suite à ces résultats, les auteurs procèdent à une comparaison proprement dite de leur métrique et des métriques existantes et cela en appliquant l'analyse de composante principale (*Principal Component Analysis* PCA), qui est une technique d'analyse de données exploratoire assez connue. Cette technique a permis aux auteurs d'identifier cinq (5) composantes principales (PC) qui décrivent 94.6% des cas. Voici les cinq (5) PC, tels que présentés par les auteurs :

- PC1 (48.2%) : les métriques LCC, TCC et Co dépendent généralement du nombre de paires de méthodes qui traitent directement ou indirectement les mêmes variables d'instance.

- PC2 (27.5%) : LCOM1, LCOM2 et LCOM3 calculent le nombre de paires de méthodes qui ont des variables d'instance communes.

- PC3 (7.9%) : LCOM5 et Coh, sont concernés par le nombre d'interaction entre les variables d'instance et les méthodes.

- PC4 (5.9%) : CBMC dépend du degré de connectivité entre les membres d'une classe en excluant les méthodes spéciales.

- PC5 (5.1%) : LCOM4, calcule le partage des variables d'instance en considérant celui fait indirectement par l'appel de méthodes.

Tel que déjà énoncé, la métrique CBMC arrive à détecter un aspect des propriétés des classes (la connectivité) que les métriques existantes ne perçoivent pas. Mais, selon les auteurs, « *cette métrique est complexe en raison de sa définition récursive et du dédale de son étape d'identification des méthodes adhésives* ».

5.2.3 Considérations et critiques

La métrique CBMC est une métrique qui considère les méthodes spéciales d'une classe comme des méthodes qui ne doivent pas influencer la cohésion de cette dernière, et adopte

comme critère de cohésion la connectivité entre les membres de cette classe. Les auteurs par cela ont essayé d'« incorporer » l'une des « caractéristiques » des classes dans la définition de leur métrique. Ce qui rend cette dernière plus appropriée à la COO par rapport aux métriques existantes.

À notre connaissance c'est le premier article qui présente une métrique de cohésion qui prend en considération les caractéristiques d'une classe OO. Et il nous pousse à nous demander quelles sont les autres caractéristiques qui méritent être considérées par les métriques C&C OO. D'autre part, les auteurs confirment qu'il existe d'autres types de méthodes spéciales, et donc nous nous demandons : quelle est la technique à utiliser pour les détecter ?

5.3 Allen et Khoshgoftaar (2001) « Mesure cu couplage et de la cohésion des modules logiciels : Une approche axée sur la théorie de l'information »

5.3.1 But

Le but de cet article est de proposer deux métriques de couplage et une de cohésion du module, fondées sur la théorie de l'information, et satisfaisant les propriétés définies par Briand, Morasca et Basili (Briand, Morasca et Basili, 1996).

5.3.2 Contenu

Dans leur précédent article qui date des années 1990, Allen et Khoshgoftaar avaient proposé d'appliquer la théorie de l'information dans les recherches futures sur les métriques pour les modules (Allen et Khoshgoftaar, 1999). Dans le présent article, ils proposent des métriques de C&C axées sur la théorie de l'information. Ces métriques sont similaires à celles présentées dans l'article précédent (Allen et Khoshgoftaar, 1999), sauf qu'elles sont appliquées au niveau du module au lieu du système. Les métriques présentées sont les suivantes :

- Couplage inter-modules (*Coupling* (mk/MS))
- Couplage-intra-module (*IntramoduleCoupling* (mk/MS))

- Cohésion du module (*Cohesion* (mk/MS)),

Où, mk représente le module sur lequel on applique la métrique et MS le système auquel appartient le module mk.

Le protocole de mesure de ces trois métriques est le même que celui des trois anciennes métriques des mêmes auteurs, c'est-à-dire les graphes d'appel. Et l'instrument de mesure se base sur les mêmes notions de théorie de l'information.

Comme les métriques de C&C du système de ces mêmes auteurs, ces nouvelles métriques vérifient les propriétés de Briand, Morasca et Basili (1996), présentées en détails dans la section V.8.2. Les graphes d'arêtes inter-modules et d'arêtes intra-module sont restés les mêmes. Mais, la différence est dans l'application de ces métriques. Par exemple, pour calculer le couplage inter-modules du système (*Coupling* (MS)), il faut considérer toutes les arêtes externes de tous les boîtiers (modules) qui lient les différents sommets ne se trouvant pas dans le même module. Alors que, pour calculer le couplage inter-modules du module mk (*Coupling* (mk/MS)), il faut considérer seulement les arêtes externes à ce module et qui lient ses sommets aux sommets des autres modules. Même chose pour le couplage Intra-module du module (*IntramoduleCoupling* (mk/MS)). Et la cohésion du module mk est calculée à partir des couplages *Coupling* (mk/MS) et *IntramoduleCoupling* (mk/MS).

Les auteurs trouvent une forte corrélation entre le couplage inter-modules du module et la cohésion du module et, respectivement, le nombre d'arêtes du module (*IntermoduleEdges*) et le ratio intra-module (*IntramoduleRatio*), qui sont des métriques de C&C basées sur le comptage.

Pour démontrer l'affinité des métriques axées sur la théorie de l'information par rapport à celles basées sur le comptage, ils comparent le couplage inter-modules du module à la mesure *IntermoduleEdges* de certains modules. Dans le cas considéré, les modules sont des fichiers sources composés d'un ensemble de fonctions codées en C qui représentent les nœuds, et les différents appels entre les fonctions sont représentés par des arêtes. Suite à cette comparaison, les auteurs remarquent que certains modules, tout en ayant le même nombre

d'arêtes inter-modules, n'ont pas les mêmes valeurs de la métrique du couplage inter-modules, ce qui prouve que leur méthode axée sur la théorie de l'information a une plus grande capacité de discrimination.

5.3.3 Considérations et critiques

Le couplage du système donne une idée générale sur les relations qui existent entre les modules de ce dernier. Ce qui est important, mais pas suffisant, car, en cas de couplage faible d'un système, on ne saura identifier la source de cette faiblesse si on n'a pas une idée sur le degré de couplage de chacun de ses modules. Même chose pour la cohésion.

Donc, passer du niveau système au niveau module (sous-système) permet de raffiner les mesures de C&C. C'est ce qui nous permet de penser que les métriques présentées dans cet article complètent les trois autres exposées dans l'ancien article des mêmes auteurs dans les années 1990. Ensemble, elles peuvent servir de métriques d'évaluation du couple C&C d'un système et de ses modules.

D'autre part, nous avons remarqué qu'une grande partie de cet article est une reprise de l'ancien. Il aurait peut-être fallu faire référence à ce dernier ou, au moins, noter quelque part ce « copiage ».

Puisque leur protocole de mesure peut se baser sur n'importe quelle abstraction du système, comme le soulignent les auteurs dans les deux articles, alors pourquoi ne pas opter pour un protocole lié à la COO et effectuer l'étude empirique sur un système OO ?

Dans leur ancienne étude, pour montrer la finesse discriminatoire de leurs métriques par rapport à d'autres métriques de C&C basées sur le comptage, ils ont comparé le couplage inter-modules du système basé sur la théorie de l'information à la métrique de couplage « nombre d'arêtes Inter-modules », et, comme ils le disent ils n'ont traité ni la cohésion ni le couplage intra-module du système. La même chose se reproduit dans cet article. Nous pensons qu'il aurait fallu présenter de nouvelles comparaisons touchant aux deux autres métriques non analysées par une étude empirique dans leur ancien travail.

Parmi les critiques que nous avons effectuées pour l'ancien article (voir section V.8.3), nous trouvons :

- Une critique concernant la signification des valeurs numériques de ces métriques, du fait que les auteurs ne donnent aucune signification aux valeurs numériques.

- Une critique de la démonstration de la haute finesse discriminatoire par rapport aux métriques de C&C basées sur le comptage, car nous avons trouvé qu'il aurait été intéressant de comparer les résultats de leur métrique, de la métrique basée sur le comptage et d'une autre métrique externe afin de montrer la finesse de leur métrique par rapport à la métrique basée sur le comptage.

Ces critiques sont de même applicables pour cet article.

5.4 Darcy et Kemerer (2002) « Complexité de logiciel : Vers une théorie unifiée du couplage et de la cohésion »

5.4.1 But

Le but de cet article est de répondre aux deux questions de recherche suivantes :

- « *Comment choisir théoriquement un petit ensemble efficace de mesures utilisables dans l'analyse de la complexité du logiciel* ? »
- « *Comment régler les conflits potentiels entre la cohésion et le couplage* ? »

5.4.2 Contenu

D'après les auteurs, la complexité du logiciel ne peut pas être capturée par une seule mesure, et il n y a pas vraiment de limites dans la façon de mesurer la complexité

En général, le développement d'un logiciel passe par deux phases majeures : (1) conception et abstraction de la solution et (2) codage de ce modèle de conception[33]. Les activités de ces deux phases se fondent sur l'approche « diviser pour régner » qui est, généralement, l'approche utilisée dans la résolution d'un problème complexe. Les auteurs

---

[33] Ici la phase concernant les exigences est « oubliée », mais cet oubli est naturel étant donné le contexte.

considèrent que les activités de développement d'un logiciel, comme d'autres types d'activités, peuvent être considérées comme des « *tâches* » analysables par « *des méthodes et des modèles d'analyse* ». Ainsi, pour répondre à leur première question de recherche, les auteurs étudient le modèle de complexité de la tâche de Wood (1986). Selon ce modèle, les tâches ont trois composants essentiels, définies[34] comme suit :

- Produit : « *entités créées ou produites par les comportements et pouvant être observées et décrites indépendamment des comportements ou des actes qui les produisent* » (Tracz, 1979). En GL, on fait correspondre « module »[35] (fonction ou classe) au terme « produit » de Wood (1986).

- Acte : « *une suite de comportements avec des but ou des directions distinctes* » (Pair, 1993). En GL, un acte est une variable de sortie d'un module.

- Indice : « *un renseignement concernant les attributs des objets qui causent les stimuli : attributs sur lesquels un individu peut baser les jugements qu'il doit porter pendant l'exécution de la tâche* » (Pair, 1993). Indice, en GL, désigne les jetons de données selon Bieman (1994) et les opérandes selon Halstead (1977).

Utilisant ces trois composants, Wood définit trois (3) sources de complexité de la tâche : complexité du composant, complexité coordinatrice, complexité dynamique. L'analyse des ces trois sources de complexité de la tâche permet aux auteurs de faire les correspondances suivantes :

- ***Complexité du composant :*** définie comme « *la fonction du nombre d'opérandes ou jetons de données qui doivent être traités lors de l'exécution de tous les actes requis pour l'accomplissement de la tâche* ». Or, puisqu'en général la cohésion est définie comme la mesure de « l'union » d'un module, les auteurs considèrent qu'au niveau du module, la cohésion correspond à la complexité du composant.

- ***Complexité coordinatrice :*** « *couvre la nature des relations entre les entrées de la tâche et les produits de la tâche* ». Or, le couplage peut être défini

---

[34] Pour définir ces composants, les auteurs se sont appuyés sur différentes références : Tracz (1979), Pair (1993), Bieman (1994) et Halstead (1977).

[35] Par souci d'uniformité avec le reste de notre étude nous appelons « module » ce que les auteurs appellent « *program unit* ».

comme la mesure du degré de liaison d'un module avec les autres modules. Ce qui permet aux auteurs de déduire que le couplage est équivalent à la complexité coordonatrice.

- ***Complexité dynamique :*** réfère aux changements de la complexité du composant et de la complexité coordinatrice durant l'exécution de la tâche. Donc elle est évaluée « *en fonction du changement de la cohésion et du couplage à travers tout le cycle de vie du logiciel* ». Mais, puisque le couple C&C fait référence à la qualité interne (statique), les auteurs considèrent que cette complexité n'est pas concernée par leur étude.

Grâce à cette mise en correspondance, les auteurs répondent à leur première question de recherche. Ils choisissent comme ensemble de mesures de la complexité logicielle les mesures du couple C&C qui donnent une idée assez claire sur « la variabilité » de la complexité structurelle du logiciel.

Les auteurs citent la grande majorité des publications traitant des métriques du couple C&C qui sont spécifiques à l'approche de programmation procédurale. Dans le tableau ci-dessous nous présentons ces travaux, tels que présentés par les auteurs et nous ajoutons une colonne (la dernière) dans laquelle nous dirons si l'article a été traité ou consulté dans notre mémoire ou non.

Tableau 5.1 Publications traitant des métriques du couple C&C citées par Darcy et Kemerer (2002)

Article	Métriques	Données empiriques	Note
Hutchens et Basili (1985)	Quatre (4) métriques de couplage basées sur le *data binding*	Deux systèmes utilitaires commerciaux	Non traité dans notre mémoire.
Fenton et Melton	Cinq (5) métriques de couplage[36]	Pas de données	Cette métrique appelée C a été abordée

[36] Les auteurs parlent de cinq métriques de couplage proposées par Fenton et Melton (1990), alors que Woodward (1993) dit que ces derniers en ont proposé une seule, nommée C. Nous pensons que cette

Article	Métriques	Données empiriques	Note
(1990)		empiriques	par Woodward (1993) et jugée subjective. L'article de ce dernier a été synthétisé dans l'Appendice A, et analysé dans la section V.4.
Adamov et Richter (1990)	Une (1) métrique de cohésion et une (1) métrique de couplage	Un utilitaire UNIX moyen	Non traité dans notre mémoire.
Tesch et Klein (1991)	Une (1) métrique de cohésion et cinq (5) autres de couplage	Système d'un manuel scolaire	Non traité dans notre mémoire.
Rising et Calliss (1992)	Huit (8) métriques de cohésion et de couplage	Un système moyen	Non traité dans notre mémoire.
Offutt, Harrold et Kolte, (1993)	Onze (11) métriques de cohésion	Deux petits et trois systèmes moyens	Non traité dans notre mémoire
Lakhotia (1993)	Sept (7) métriques de cohésion[37]	Pas de données empiriques	Article synthétisé dans l'Appendice A et analysé dans la section V.2..
Bieman et Ott (1994)	Une (1) mesure de cohésion	Cinq échantillons de procédures on été analysés	Cette métrique a été traitée dans l'Article de Bieman et Kang (1998), que nous avons synthétisé dans l'Appendice A et analysé dans la section V.6. En fait, c'est une famille de trois métriques de cohésion.

À partir du Tableau 5.1, les auteurs remarquent que « *la majorité de ces travaux sont de nature théorique et observationnelle, avec seulement un minimum de travail empirique pour la validation des métriques qu'ils proposent* ».

---

métrique détermine le couplage d'une paire de modules parmi cinq types. C'est pour cette raison que nous considérons qu'il s'agit de cinq (5) métriques.

[37] En fait il s'agit d'une seule métrique de cohésion basée sur « le principe d'association ». Mais les auteurs ici parlent de sept (7) métriques car le principe d'association détermine le degré de cohésion d'un module parmi les sept niveaux de cohésion de Yourdon et Constantine (1979).

D'autre part, les auteurs partagent l'avis de Chidamber et Kemerer (1994) et de bien d'autres à propos du besoin de métriques spécifiques à la COO, parce qu'ils trouvent que tous les travaux concernant le couple C&C dépendent uniquement des idées et de la terminologie de la conception procédurale. De plus, ils considèrent que ces travaux manquent de base théorique solide.

Les auteurs se sont intéressés particulièrement à l'étude de Chidamber et Kemerer (1994) dans laquelle ces derniers proposent six métriques de la complexité du logiciel OO (appelées suite CK). Les auteurs considèrent la métrique LCOM de la suite CK comme une métrique de cohésion et deux autres métriques (RFC et CBO) comme des métriques de couplage. D'après les auteurs, cette suite a constitué une source originale pour l'invention de plusieurs métriques OO, comme celles proposées dans l'étude de Briand, Daly et Wust (1999) comprenant treize (13) métriques de cohésion et trente (30) métriques de couplage OO.

En ce qui concerne la deuxième question de recherche de Darcy et Kemerer (2002), à savoir « *comment régler le conflit qui a toujours existé entre le couple C&C, de façon à choisir un niveau acceptable pour chacun d'entre eux* ?», les auteurs emploient les concepts de cognition (mémoire à court terme, mémoire à long terme et fragments) et de complexité de la tâche comme fondement théorique pour y répondre.

Par exemple, un module qui a zéro (0) couplage avec les autres modules demande un « fragment de compréhension ». S'il est couplé à un autre module, sa compréhension demande un autre fragment ajouté à la pile. S'il est par contre couplé à deux modules, alors sa compréhension demande plusieurs fragments de compréhension (trois ou plus). Donc, la taille de « la pile de compréhension » d'un module dépend, en premier lieu, du degré de couplage de ce dernier, et, il ne serait question de cohésion que si toutes les relations de couplage avec les autres modules sont claires dans la tête du programmeur. Cette réflexion mène les auteurs à faire la proposition suivante :

P1 : Pour des modules fortement couplés, la performance de compréhension diminue.

À partir de ça on confirme que c'est le couplage qui guide un programmeur dans la compréhension d'un module car la performance de compréhension est influencée par le degré de couplage de ce module aux autres modules. Dans tous les travaux qui portent sur le couple C&C, les auteurs ont remarqué une chose étonnante : « *les concepts de C&C ainsi que leurs mesures sont normalement étudiés séparément. Alors que leurs effets sur la complexité sont interdépendants. Dans la littérature on parle souvent d'effets de la cohésion quand il faudrait plutôt parler d'effets conjoints du couple C&C. Et l'impact de la cohésion le plus connu en littérature est commun à celui de couplage* ». À partir de cela les auteurs présentent cette proposition :

P2 : La compréhension des modules fortement couplés est améliorée lorsque l'on est en présence d'une forte cohésion.

Pour terminer avec la deuxième question, c'est-à-dire régler le problème de conflit entre le couple C&C dans n'importe quelle approche et dans n'importe quel langage de programmation, ils proposent :

P3 : L'impact du couple C&C sur la compréhension d'un module n'est nullement influencé par le langage de programmation.

5.4.3 Considérations et critiques

C'est un article très intéressant car il aborde une question de recherche délicate qui est de savoir comment trouver le bon équilibre entre C&C de façon à ce qu'ils soient acceptables. Cette question a toujours existé mais, à notre connaissance, elle n'a jamais été soulevée. La réponse que proposent les auteurs se résume en trois propositions (P1, P2 et P3 citées ci-haut) que nous appuyons.

5.5 Meyers et Binkley (2004) « Métriques de la cohésion basées sur le tranchage et intervetion du logiciel »

5.5.1 But

Présenter une étude empirique, assez étendue, de cinq métriques de cohésion fondées sur la méthode de tranchage, parmi les six proposées auparavant par Ott et Thuss (1993).

5.5.2 Contenu

Les auteurs considèrent que leur article apporte trois contributions importantes à la qualité logicielle basée sur la méthode de tranchage :

1. Les mesures de cohésion fondées sur le tranchage, à cause de leur corrélation avec la qualité, peuvent être employées « *pour guider et mesurer les efforts d'intervention sur le logiciel* ».
2. Les valeurs pour les cinq (5) métriques considérées « *sont utiles pour identifier les modules dégradés* ».
3. Une comparaison des métriques montre « *quelles métriques sont fortement corrélées et lesquelles donnent des visions différentes des programmes* ».

Les cinq métriques analysées sont : *Coverage, MaxCoverage, MinCoverage, Overlap*, et *Tightness*. La métrique *Parallelism* ne sera pas traitée car elle est jugée complexe et l'information qu'elle fournit est donnée par quelques unes des cinq métriques étudiées et qui sont plus faciles à comprendre. Ces métriques sont définies dans l'Appendice A.

Cette étude est faite sur un logiciel de 1.1 million lignes de code, qui compte 22,651 procédures de 63 programmes. En tout 2,067,016 tranches sont analysées. Les auteurs considèrent une procédure comme un module – un module étant l'unité du code sur laquelle ils appliquent les métriques. Ils appliquent les cinq métriques sur les 22,651 modules du logiciel et présentent les résultats sous formes de :

- Tableau, qui pour chaque métrique fournit la moyenne, l'écart type et l'intervalle de confiance[38].

---

[38] « Intervalle construit autour de la valeur observée à partir d'un échantillon, et ayant une certaine probabilité de contenir la valeur réelle de la caractéristique étudiée. L'écart entre la borne inférieure et

- Cinq diagrammes pour les cinq métriques étudiées. Chacun représente les moyennes de la métrique correspondante pour chaque programme (ensemble de modules).

À partir du tableau les auteurs observent que la métrique *Overlap* a l'écart type le plus élevé. Ce qui montre qu'il s'agit d'une métrique très sensible, contrairement à la métrique *MaxCoverage* qui paraît être la métrique la moins sensible parmi celles étudiées. Ceci implique que « les valeurs de *MaxCoverage* sont plus significatives que celles d'*Overlap* ».

À partir des cinq diagrammes, les auteurs remarquent que :

- La grande variation des moyennes des métriques *Tightness*, *MinCoverage* et *Overlap* : ces dernières peuvent donc fournir une bonne discrimination parmi les programmes.

- Toutes les mesures augmentent quand *Tightness* augmente.

- *Coverage*, *MaxCoverage* et *Overlap* sont sensibles aux changements de *Tightness*, contrairement à la métrique *MinCoverage* qui ne manifeste qu'une sensibilité très faible par rapport à cette métrique.

- *Overlap* a un comportement particulier surtout si on la compare à *Coverage* et *MaxCoverage*.

Afin d'examiner la sensibilité des métriques au « vieillissement » des modules, les auteurs procèdent à une étude temporelle de deux modules. Suite à cette analyse, ils remarquent :

  - Ces métriques ont la capacité de « quantifier la détérioration » de la cohésion que normalement un module subit avec l'âge. Les auteurs proposent d'adopter cette démarche dans la validation de futures métriques de cohésion.

  - Toutes les métriques étudiées, exceptée la métrique *Overlap*, ont une forte corrélation positive. Leurs valeurs sont élevées pour la première et la deuxième version mais à partir de la troisième elles régressent.

---

la borne supérieure de l'intervalle est égal à la valeur observée plus ou moins la marge d'erreur. » (OQLF, 2008)

Et pour mieux comprendre les relations entre ces métriques, les auteurs font une comparaison de chaque paire de métriques via le coefficient de corrélation de *Pearson*. Ils identifient les relations suivantes :

- Une corrélation forte entre *MinCoverage*, *Tightness* et *Overlap*.
- *MaxCoverage* est la métrique la plus faiblement corrélée avec les autres métriques.
- *Overlap* est la métrique la plus variable, car elle montre une corrélation avec *MinCoverage* et *Tightness* (premier résultat de cette comparaison) et, d'autre part, ne montre aucune relation linéaire avec *Coverage* et *MaxCoverage*. Donc, la métrique *Overlap* peut fournir une vue sur le programme tout à fait différente à celle donnée par *MaxCoverage*.

5.5.3 Considérations et critiques

Ce travail est une sorte de continuité de l'article d'Ott et Thuss (1993), que nous avons synthétisé dans l'Appendice A et analysé dans la section V.3. Ott et Thuss ont proposé un ensemble de métriques de cohésion basées sur le tranchage. Parmi les points faibles dont souffrait l'étude de ces derniers, nous avons remarqué la petite dimension des modules utilisés pour montrer l'efficacité de ces métriques. Le présent travail a corrigé cette faiblesse en appliquant ces métriques sur un logiciel qui compte 1.1 millions de lignes de code.

Dans le précédent travail, Ott et Thuss (1993) ont fourni un tableau dans lequel ils ont fait correspondre une valeur ou un intervalle de valeurs numériques de ces métriques à trois niveaux de cohésion (forte, acceptable et faible), mais sans fournir aucune information sur les relations qui peuvent exister entre ces métriques. Cet article comble cette lacune. Une comparaison quantitative et qualitative de chaque paire de métriques nous permet de savoir quelles sont les métriques corrélées, et lesquelles fournissent des résultats plus significatifs que les autres.

Lorsqu'ils abordent les métriques basées sur le tranchage présentées par Weiser (1981), les auteurs disent que ces métriques ont été proposées par ce dernier comme mesure de cohésion et de complexité. Or, en lisant l'article de Weiser (1981), nous ne trouvons

même pas le terme cohésion, ni un de ses synonymes : les métriques étaient proposées comme des métriques de complexité.

Par leur étude comparative, les auteurs ont déduit que quelques-unes des métriques étudiées donnent des visions différentes par rapport à d'autres, puisque le but de toutes ces métriques est de mesurer la cohésion. Donc, nous nous demandons : quelles sont ces visions ? Et que représentent-elles pour la mesure de la cohésion ?

5.6 Yang et Berrigan (2005) « Détection du couplage indirect »

5.6.1 But

L'objectif principal de cet article est de démontrer qu'il existe des formes de fermetures transitives[39] de couplage que l'on ne peut pas réduire à des couplages directs et que « *ces formes* [de couplage] *sont une source importante d'erreurs dans le développement logiciel* ».

5.6.2 Contenu

Les auteurs s'intéressent à l'attribut de qualité du logiciel « couplage » parce qu'ils croient qu'il est utile comme indicateur de la qualité de conception et qu'il sert d'identificateur d'anomalies potentielles, qui risquent d'affecter le coût de développement et de maintenance. D'après les auteurs cet attribut de qualité est resté ambigu car, dans sa définition, il y a un emploi de termes informels comme « force » et « interconnexion ».

Selon les auteurs, Chidamber et Kemerer (1994) sont les premiers chercheurs qui ont considéré le couplage d'un logiciel OO, suivis par Briand, Daly et Wust (1999), qui ont présenté un cadre pour la mesure du couplage dans les systèmes OO, dans lequel ils ont décrit environ trente (30) métriques de couplage dont seulement deux (2) détectent certaines formes de couplage indirect.

---

[39] Informellement : si A est couplé avec B et B avec C alors la fermeture transitive implique l'introduction d'un couplage indirect entre A et C.

Briand, Daly et Wust (1999) définissent le couplage direct comme « *quelque chose qui décrit une relation entre un ensemble d'éléments* » (Briand, Daly et Wust, 1999), et le couplage indirect comme « *la fermeture transitive de la relation décrite par le couplage direct* » (Briand, Daly et Wust, 1999). Selon les auteurs la définition du couplage indirect admet deux interprétations :

- Si on ne considère pas la fermeture transitive de toutes les relations de couplage, on ne capture pas toutes les formes de couplage. Ce qui implique que cette définition est incomplète.

- Si on considère les fermetures transitives de toutes les relations de couplage, donc toutes les classes sont couplées les unes aux autres. Ce qui implique que cette définition est inutile.

Ainsi, cette définition est ouverte à ces deux mauvaises interprétations parce que Briand, Daly et Wust (1999) n'ont pas précisé quelles sont les relations à considérer et comment doivent être ces relations. En effet, les fermetures transitives de toutes les relations de couplage existantes entre les classes d'un système consistent en la liaison de tous les couplages directs de ces classes, rendant toutes ces dernières couplées les unes aux autres. Ce qui est un résultat d'aucun intérêt. Un exemple illustrant ce problème est présenté dans l'Appendice A.

Pour résoudre ce problème, les auteurs pensent qu'il est essentiel de donner une définition formelle au terme « couplage ». Pour cela, ils adoptent un nouveau critère d'identification du couplage entre deux modules, qui se résume à la question posée par Yourdon et Constantine (1979) : « *jusqu'à quel point doi-ont connaitre un module pour comprendre un autre module* ? » et définissent le couplage indirect comme : « *n'importe quelle relation de couplage qui n'est pas un couplage direct* ». Enfin, ils proposent pour déterminer le couplage direct entre deux modules, une analyse superficielle qui permet de détecter facilement les relations entre deux modules. Les relations qui nécessitent une analyse assez détaillée et approfondie pour être détectées sont identifiées comme des relations de couplage indirect (Yang et Berrigan, 2005).

Les auteurs définissent deux types de couplage indirect :

- Couplage transitif simple (*Simple Transitive Coupling STC*), « *un couplage indirect identifié par la fermeture transitive d'une seule relation* ».
- Couplage transitif composé (*Composit Transitive Coupling CTC*), « *un couplage indirect identifié par la fermeture transitive de la composition de plus d'une relation* ».

Parmi les buts de cet article, il y a la conception d'un outil de détection de couplage indirect. Cet outil, nommé détecteur de couplage indirect (*Indirect Coupling Detector ICD*), est un « plugiciel » d'Éclipse.

Le programme d'ICD traduit un algorithme présenté et expliqué dans l'Appendice A. Cet algorithme détecte une forme particulière de couplage indirect appelée « couplage indirect *use-def* ». Ce type de couplage est identifié lorsque deux classes sont connectées par une chaîne de valeur *use-def*. « *La définition de cette valeur est retournée vers le haut de la chaîne d'appel, puis passée en bas où elle doit être employée* ». Un exemple de ce type de couplage est présenté dans l'Appendice A. L'architecture de l'outil ICD est de même présentée dans l'Appendice A.

Enfin, les auteurs ont appliqué ICD sur plusieurs projets existants d'à peu près 25 000 lignes de code, et 130 classes et interfaces. Ils ont jugé que les résultats de cette expérimentation étaient intéressants puisque les couplages indirects *use-def* ont bien été détectés par l'outil.

5.6.3 Considérations et critiques

Dans ce travail, Yang et Berrigan (2005) critiquent et corrigent la vision des chercheurs sur le couplage indirect – une vision qui considère n'importe quelle forme de couplage indirect entre deux modules comme « *une simple fermeture transitive d'une forme de couplage direct* », alors qu'il existe des formes de couplage indirect plus complexes.

Les auteurs ont présenté trois types de couplage indirect : couplage indirect STC, couplage indirect CTC et couplage indirect *use-def*. Mais nous ne savons pas si ce sont les seules formes de couplage indirect que les auteurs ont pu déceler ou s'il en existe d'autres. Ce point n'a pas été traité explicitement dans cet article. Mais, d'après les résultats de leur expérimentation, nous avons pu deviner que les formes de couplage indirect ne se restreignent pas aux trois types abordés par les auteurs, puisque dans leur expérimentation, les auteurs ont découvert une nouvelle forme de couplage indirect.

Voici un exemple qui illustre cette nouvelle forme de couplage indirect : soit une classe X ayant un couplage indirect *use-def* avec une classe Y, ayant elle-même un couplage indirect *use-def* avec une classe Z. Une interaction entre les deux chaînes *use-def*, implique une forme de couplage indirect entre X et Z.

D'autre part leur outil ne détecte que le couplage indirect *use-def*. Pourquoi cet outil ne détecte pas les couplages indirects STC et CTC ? Nous ne jugeons pas que c'est une limite de l'outil, mais nous ne savons pas pourquoi les auteurs ont choisi d'automatiser la détection d'une forme de couplage et non pas les autres.

Nous considérons ce travail comme une ébauche de l'étude et de l'analyse détaillé du couplage indirect, car il y a encore beaucoup de questions de recherche qui nécessitent des réponses, comme par exemple :

- Est-ce qu'il y a d'autres formes de couplage indirect à part celles abordées par les auteurs ?
- Peut-on automatiser la recherche des couplages indirects STC et CTC comme les auteurs l'ont fait pour le couplage indirect *use-def* ?
- Y a-t-il vraiment un impact du couplage indirect sur des attributs de qualité comme la maintenabilité et la modifiabilité ?
- Afin d'éviter les problèmes que peut causer le couplage indirect, à quelle phase du développement logiciel serait-il intéressant de le détecter ?

- Peut-on concevoir des mesures de couplage indirect (à part la détection[40]) ?

5.7 Kaung, Khan et Thein (2005) « Pour visualiser le couplage entre les modules »

5.7.1 But

Le but de cet article est de proposer huit (8) nouvelles métriques de couplage, et de créer un système qui détecte ces huit formes de couplage dans un programme donné.

5.7.2 Contenu

Les auteurs considèrent les mesures de couplage comme des indicateurs des attributs de la qualité logicielle suivants : la facilité de compréhension, la facilité des tests, la maintenabilité et la fiabilité.

Ils invoquent les trois (3) raisons données par Page-Jones (1980) qui incitent les concepteurs à concevoir des modules à couplage faible car cela :

1. « *réduit la chance qu'un défaut dans un module cause un échec (ou une défaillance) dans d'autres modules,*
2. *[...] réduit la chance que des changements apportées à un module causent des problèmes à d'autres modules, ce qui augmente la réutilisabilité, et*
3. *[...] réduit le temps nécessaire à un programmeur pour les comprendre* ».

Les auteurs croient que la visualisation du couplage peut apporter plus que sa quantification numérique, qui a été le sujet de plusieurs travaux traitant du couplage. Ils considèrent un module comme une simple fonction ou procédure, et définissent le couplage comme « *la mesure de l'interdépendance entre deux modules logiciels* ».

Selon la direction d'un message s'échangeant entre deux modules, les auteurs différencient deux types de couplage :

---

[40] Les auteurs comme Fenton et Pfleeger (1997) voient la détection comme une forme de mesure (Yang et Berrigan, 2005).

- Couplage d'exportation : « *compte les messages envoyés à partir du module* ».
- Couplage d'importation : « *compte les messages reçus par le module* ».

À partir de la direction du message et du type de données contenues dans ce dernier, huit métriques (ou directions) de couplage sont proposées par les auteurs. Dans ce qui suit, nous allons les présenter telles que définies dans l'article :

1. « ***Couplage d'importation d'appel :*** *un module appelle un autre module, mais il n'y a aucun paramètre ou variables de référence communs.*
2. ***Couplage d'importation scalaire :*** *une variable scalaire dans un autre module est passée comme paramètre effectif au module appelé A.*
3. ***Couplage d'importation de timbre :*** *un enregistrement dans un autre module est passé comme paramètre effectif au module appelé A.*
4. ***Couplage d'importation de « tramp » :*** *un paramètre formel dans un autre module est passé comme paramètre effectif au module B. Le module B passe ensuite le paramètre formel au module A, sans que B ait accédé ou modifié la variable.*
5. ***Couplage d'exportation d'appel :*** *un module appelle un autre module, mais il n'y a aucun paramètre ou variables de référence communs.*
6. ***Couplage d'exportation scalaire :*** *une variable scalaire dans le module A est passée comme paramètre effectif à un autre module.*
7. ***Couplage d'exportation de timbre :*** *un enregistrement dans le module A est passé comme paramètre effectif à un autre module.*
8. ***Couplage d'exportation de tramp :*** *un paramètre formel dans le module A est passé comme paramètre effectif au module B. Le module B passe ensuite le paramètre formel correspondant à un autre module sans qu'il ait accédé ou modifié la variable* ».

Les auteurs ont développé un système qui détermine le couplage d'un programme C parmi les huit (8) types de couplage qu'on vient de définir. Ce système comprend trois (3) composants : analyseur syntaxique, analyseur de couplage et visualiseur (visionneur) de couplage. La description de chacun de ces composants est détaillée dans l'Appendice A.

Les auteurs appliquent leur système pour visualiser les directions de couplage qui existent entre les procédures d'un ensemble de quatre (4) programmes. Ces programmes comptent de 45 à 75 procédures et de 1360 à 2500 lignes de code.

5.7.3 Considérations et critiques

Kaung, Kham et Thein voient le couplage entre deux modules comme un échange de message. Et, à partir de l'information contenue dans ce message, ils déterminent le type de couplage : couplage *tramp*, couplage timbre, couplage scalaire ou couplage d'appel. Par la suite, en associant la direction du message – c'est-à-dire message qui sort du module ou qui entre dans le module – ils définissent huit métriques ou directions de couplage. Mais, parmi ces huit métriques, ils ne différencient pas le couplage le plus fort du couplage le plus faible.

Les auteurs jugent que la visualisation du couplage apporte plus que sa quantification et ont eu comme objectif, à travers leur système, de visualiser les différentes formes de couplage. Seulement, nous ne voyons dans leur article aucune forme de visualisation du couplage à part un tableau. Nous nous demandons si le titre de l'article n'est pas abusif.

Nous avons de même été déçus du fait que, même si cet article a été publié en 2005, il traite le couplage selon l'approche de la programmation procédurale.

5.8 Atole et Kale (2006) « L'évaluation des principes de couplage et cohésion de paquet pour la prédiction de la qualité de conception orientée objet »

5.8.1 But

Le but de cet article est de pronostiquer la qualité de conception, dans une phase avancée du cycle de vie d'une application OO, via la mesure du couple C&C.

5.8.2 Contenu

Les auteurs évoquent le problème de l'ambiguïté des définitions du couple C&C. Ils considèrent la cohésion au niveau d'une classe comme « *l'un des attributs les plus importants*

d'un logiciel OO » et la définissent comme « *la force des relations entre les méthodes d'une classe* ».

Toujours d'après les auteurs, le couplage en OO est vu comme un attribut d'un couple de classes plutôt que du système en entier. On dit qu'un objet A est couplé à un objet B si A envoie un message à B. En OO, on énumère deux sources de couplage entre deux classes :

- « *Une classe qui en contient une autre.*
- *L'interface des méthodes d'une classe prend comme paramètres les instances d'autres classes.* »

La décomposition en classes est nécessaire mais pas suffisante dans « l'organisation d'une conception OO ». Donc, selon les auteurs, il paraît utile de mesurer ces deux concepts au niveau des *paquets*[41]. Trois principes de cohésion de paquet ont été choisis :

1. **Le principe d'équivalence des versions (*Release Equivalent Principe REP*)** : Un élément réutilisable, que ce soit un composant ou un ensemble de classes, ne peut pas être réutilisé à moins qu'il soit identifié par un système de version.

2. **Le principe de fermeture commune (*Common Closure Principle CCP*)** : Des classes qui changent ensemble appartiennent au même groupe. Quand nous groupons les classes qui changent ensemble dans les mêmes *paquets*, alors l'impact du paquet d'une version à une autre sera réduit au minimum.

3. **Le principe de réutilisation commun (*Common Reuse Principle CRP*) :** des classes qui ne sont pas employées ensemble ne devraient pas être groupées ensemble.

Pour le couplage, trois principes de couplage de paquets parmi ceux existants ont été sélectionnés :

---

[41] Les auteurs ne disent pas explicitement ce qu'est un « *paquet* ». Mais, à partir des figures contenues dans l'article, nous devinons qu'ils considèrent un *paquet* comme un contenant d'un ensemble de classes.

1. **Le principe des dépendances acycliques (*Acyclic Dependencies Principle ADP*) :** « *Les dépendances entre les paquets ne doivent pas former des cycles.* »

2. **Le principe des dépendances stables (*Stable Dependencies Principle SDP*) :** « *La stabilité est liée à la quantité de travail exigée pour apporter une modification. Une manière sûre de rendre un paquet stable, et donc difficile à modifier, c'est de faire en sorte que d'autres paquets dépendent de lui.*».

3. **Le principe d'abstraction stable (*Stable Abstraction Principle SAP*) :** « *les paquets stables devraient être des paquets abstraits.* »

La métrique de cohésion choisie par les auteurs est la métrique de cohésion relationnelle H proposée par Martin (1997), dont la formule est :

$$H = (R + 1)/N$$

Avec :

- R : le nombre de relations entre les classes du paquet dont on mesure le degré de cohésion, et
- N : le nombre de classes contenues dans ce paquet.

En ce qui concerne le couplage entre paquets, Martin (2000) propose quelques métriques pour caractériser le couplage.

L'instabilité[42] d'un paquet donne une idée sur son indépendance par rapport à d'autres paquets, de telle sorte que plus le paquet est instable, moins il dépend des paquets auxquels il est lié. L'instabilité (I) d'un paquet est fonction du nombre de dépendances entrantes et du nombre de dépendances sortantes.

$$I = (Ce/ (Ca + Ce))$$

---

[42] Il est clair que le terme stabilité désigne quelque chose de positif en GL, sauf que dans cet article, ce que les auteurs veulent dire par stabilité c'est « la rigidité » d'un paquet et la difficulté de lui apporter des modifications quand il le faut. Ce qui lui confère une connotation négative.

Avec :

- Ce : le nombre de classes se trouvant à l'extérieur du paquet et dont dépendent les classes à l'intérieur de ce paquet (dépendances sortantes).
- Ca : le nombre de classes externes au paquet et qui dépendent des classes du paquet (dépendances entrantes), dans le cas où Ca = 0, I = 1, donc le paquet est instable, et il est stable quand Ce = 0.

Les paquets hautement stables sont difficiles à modifier, mais ne doivent pas être difficiles à étendre ou à prolonger. En fait, ceci est possible quand ils sont hautement abstraits. Donc, « *les paquets qui sont très dépendants les uns des autres (stables) doivent être le plus abstraits possible* ».

La formule de la métrique d'abstraction A adoptée par les auteurs est la suivante :

$$A = (Na/Nc)$$

Avec :

- Na : nombre de classes abstraites dans le paquet.
- Nc : nombre de classes du paquet.

À partir des principes et des métriques de stabilité et d'abstraction, les auteurs arrivent à décrire une structure convenable d'une application OO. En effet, c'est une « *application composée de paquets concrets, instables et faciles à modifier, et de paquets abstraits, stables et faciles à prolonger* ». Les auteurs ont remarqué une relation évidente entre la métrique d'abstraction A et celle d'instabilité I : lorsque A s'accroît, I se dégrade. Cette relation est présentée par le graphe A-I qui se trouve dans l'Appendice A avec une description détaillée de ses zones.

Les auteurs mesurent aussi la distance entre un paquet donné et la séquence principale à l'aide d'une métrique, appelée métrique de distance (D). La séquence principale est la ligne, dans le graphe A-I, sur laquelle on retrouve les paquets de bonne conception. Cette ligne

présente une sorte d'équilibre entre l'instabilité et l'abstraction du paquet, c'est-à-dire un équilibre entre les dépendances entrantes et sortantes de ce paquet (voir Appendice A). La métrique de distance, D, est définie comme suit :

$$D= |A + I - 1| / \sqrt{2}$$

La métrique plus commode et normalisée de la distance (D'), est calculée comme suit:

$$D'= |A + I - 1|$$

Quand cette métrique vaut 0, le paquet est dans la séquence principale, et quand elle vaut 1, cela veut dire qu'il est loin de la ligne de séquence principale.

Les auteurs appliquent les métriques H, I, A et D' sur un ensemble de paquets d'un système, durant la phase de conception. Cet exemple est tiré d'un livre qui apprend les bonnes pratiques de conception. Donc, ces *paquets* doivent être de cohésion forte et de couplage faible. C'est exactement ce que concluent les auteurs par les résultats de leurs métriques.

5.8.3 Considérations et critiques

Atole et Kale (2006) adoptent l'approche de Martin (2000) qui propose de mesurer le couple C&C des *paquets* d'un système au lieu des classes. C'est la première fois que nous traitons un article ayant un tel but.

La mesure du couple C&C au niveau des *paquets* permet, selon les auteurs, d'améliorer l'organisation d'un système OO. Les métriques proposées sont appliquées durant la phase de conception pour évaluer le couple C&C. Ce qui va permettre, par la suite, de prévoir la qualité logicielle.

Tel que déjà énoncé, dans leur étude de cas, les auteurs appliquent les métriques sur des *paquets* produits lors de la phase de conception. Seulement, nous ne connaissons pas exactement la nature des informations extraites de ces *paquets* qui a été utilisée dans le calcul

des métriques, ni le genre de représentation de cette conception (graphes, interface, pseudo-code, etc.). Nous reprochons donc aux auteurs le manque de détails.

D'autre part, lorsque les auteurs ont parlé de modules capables d'être étendus ou prolongés, ils n'ont pas expliqué ce qu'ils veulent dire par cela.

5.9 Conclusion

La plupart des travaux traitant du couple C&C des années 2000 que nous avons synthétisés et analysés sont fondés sur les travaux des années 1990 et sont venus dans le but de :

1. Compléter ces anciens travaux, comme c'est le cas pour :
    - l'article d'Allen et Khoshgoftaar (2001), dans lequel les auteurs ont présenté des métriques de C&C du module dérivées de celles qu'ils avaient présenté en 1999 (Allen et Khoshgoftaar, 1999).
    - l'article de Meyers et Binkley (2004), dans lequel les auteurs proposent des métriques de cohésion basées sur le tranchage, proposées dans les années 1990 par Ott et Thuss (1993), et qu'ils appliquent sur des modules d'un système de grande taille. L'étude comparative de cet ensemble de métriques leur permet de déterminer : la métrique la plus variable parmi ce groupe de métriques, les métriques qui sont en corrélation positive et les métriques qui fournissent des visions tout à fait différentes.
    - l'article d'Atole et Kale (2006), où les auteurs se fondent sur l'ancien travail de Martin (1997) où ce dernier propose des métriques de C&C appliquées sur des paquets au lieu de classes. Atole et Kale adoptent ces mêmes métriques, mais les utilisent pendant la phase conception.
2. Les critiquer en montrant leurs points faibles et apporter les corrections nécessaires que ce soit à la vision ou à la méthode. C'est le cas de :
    - l'article de Chae, Kwon et Bae (2000), dans lequel ils critiquent toutes les métriques de cohésion OO existantes (Chidamber et Kemerer, 1994) (Briand, Daly et Wust, 1998), et démontrent leur faiblesse due surtout à un manque de catégorisation des méthodes.

- l'article de Yang et Berrigan (2005), dans lequel les auteurs critiquent la définition donnée par Briand, Daly et Wust (1999) du terme « couplage indirect » et montrent comment cette définition est ouverte à de fausses interprétations.

3. Les analyser, tout en en critiquant quelques-uns et en en appuyant d'autres. C'est le cas de l'article de Darcy et Kemerer (2002). Les auteurs de cet article critiquent la majorité des anciens travaux menés sur le couple C&C, du fait qu'ils sont de nature théorique et que les études empiriques sont modestes. Et, d'autre part, ils appuient Chidamber et Kemerer (1994) dans l'idée de la nécessité de définition de nouvelles métriques de C&C propres au paradigme OO.

Parmi tous les articles des années 1990, trois (3) ont abordé des difficultés tout à fait nouvelles pour nous :

- La première concerne la cohésion. Chae, Kwon et Bae (2000) ont remarqué que toutes les principales métriques de cohésion existantes ne prennent pas en considération certaines caractéristiques des classes OO sur lesquelles elles sont appliquées. En effet, ces métriques tiennent compte de toutes les méthodes d'une classe, alors qu'il se peut que cette dernière comprenne des méthodes qui n'affectent ni « le comportement » ni « l'état » de ses objets. Chae, Kwon et Bae nomment ces méthodes « méthodes spéciales ». En outre, elles (les métriques de cohésion existantes) reposent sur le simple critère de comptage des interactions entre les membres d'une classe. Pour remédier à ce problème, Chae, Kwon et Bae ont proposé une nouvelle métrique de cohésion CBMC, qui, dans son processus de calcul, identifie les méthodes spéciales qui ne sont pas considérées par la suite, et qui adopte comme critère de mesurage « la connectivité entre les membres d'une classe ».

- La deuxième est liée aux conflits potentiels dans le couple C&C. En 2002, Darcy et Kemerer (2002) ont choisi le couple C&C comme un ensemble restreint de mesures efficaces de la complexité logicielle. Ce choix a été appuyé par une étude théorique du modèle de complexité de la tâche de Wood (1986) du point de vue du GL, et a amené les auteurs à régler les conflits qui existent entre le couple C&C, à savoir, le niveau acceptable de chacun d'entre eux et lequel des deux concepts guide la compréhension d'un module. Comme solution Darcy et Kemerer (2002) présentent trois (3) propositions :

    - *Proposition 1 :* Pour des modules fortement couplés, la performance de compréhension diminue.

- *Proposition 2 :* La compréhension des modules fortement couplés est améliorée lorsque l'on est en présence d'une forte cohésion.

- *Proposition 3 :* L'impact du couple C&C sur la compréhension d'un module n'est nullement influencé par le langage de programmation.

- La troisième a un rapport avec le couplage. En 2005, Yang et Berrigan (2005) ont analysé la définition du couplage indirect donnée par Briand, Daly et Wust (1999). Ils ont trouvé que cette dernière est ouverte à deux interprétations qui la rendent incomplète et inutile, et cela à cause du terme « relation » qui admet plus qu'une signification. Donc, l'unique solution à ce problème était d'apporter une définition formelle au couplage indirect. En effet, Yang et Berrigan (2005) définissent ce dernier comme « *n'importe quel couplage qui n'est pas un couplage direct* ». Cette définition les pousse alors à présenter une définition formelle du concept « couplage ».

Par ailleurs, un article parmi ceux étudiés, celui d'Atole et Kale (2006), fournit une nouvelle vision de la mesure du couple C&C en OO, différente de toutes celles présentées auparavant. En effet, depuis les années 1990, toutes les mesures de C&C en OO proposées ont été destinées à l'entité « classe ». Donc, Atole et Kale (2006) définissent des métriques de C&C en OO applicables au niveau « paquet » qui, selon la vision d'UML, est un contenant (ou un conteneur) d'un ensemble de classes, de telle sorte que la cohésion d'un paquet, est définie comme la force de liaison des classes dans ce paquet, et le couplage d'un couple de paquets est vu comme le degré de dépendance de ces deux paquets.

Concernant les articles d'Allen et Khoshgoftaar (2001) et de Kaung, Kham et Thein (2005), les auteurs proposent des métriques du couple C&C prédictives puisqu'elles sont appliquées lors de la conception. Donc, le besoin de concevoir des métriques qui évaluent la complexité structurelle d'un logiciel dans le but d'en prédire la qualité est toujours présent.

Dans cette décennie, les deux articles à notre avis qui sont les plus importants parmi ceux synthétisés sont :

- L'article de Darcy et Kemerer (2002). Nous jugeons qu'il est d'une importance primordiale car ses auteurs ont effectué une étude bibliographique poussée sur le couple C&C.

- L'article de Chae, Kwon et Bae (2000), qui est un article très intéressant, où les auteurs critiquent toutes les métriques de cohésion existantes qui ne prennent pas en considération les caractéristiques ou les propriétés d'une classe OO, et critiquent aussi le critère de mesurage sur lesquelles elles reposent.

CONCLUSION

C.1 Survol des trois décennies

Dans ce travail de recherche, nous nous sommes intéressés à une étude bibliographique détaillée du couple C&C. Depuis son apparition jusqu'aujourd'hui, ce couple de concepts est passé par plusieurs étapes. Dans ce qui suit, nous résumons les réalisations qui ont le plus marqué cet itinéraire :

- Début des années 1970 : établissement des définitions et des listes des types de C&C.

- Fin des années 1970 : ajout de certains types de C&C aux anciennes listes et attribution d'une valeur numérique à chaque niveau de C&C, en attirant l'attention des concepteurs sur le danger de donner trop d'importance à ces nombres.

- Début des années 1980 : critiques des définitions et des listes de catégorisation existantes à cause de leur subjectivité. Ce qui a entraîné la définition de métriques de cohésion objectives.

- Fin des années 1980 : mise en évidence de l'importance du couple C&C par l'étude de son impact sur certains attributs de la qualité logicielle.

- Début des années 1990 : apport de définitions formelles aux termes employés dans les définitions originales du couple C&C, et propositions de nouvelles métriques objectives.

- Fin des années 1990 : critiques des listes de catégorisation originales, qui reposent sur la terminologie du paradigme de programmation procédurale, et critiques des bases théoriques. D'où le besoin de définir des métriques de C&C théoriquement solides et qui sont spécifiques au paradigme OO. Cette période a de même été caractérisée par la définition de métriques de C&C prédictives, applicables durant la phase de conception.

- Début des années 2000 : critiques des métriques de cohésion OO existantes à cause de leur critère de mesure et de la non considération des propriétés de la classe OO. D'où la définition de métriques qui règlent ces problèmes. Cette période est de même caractérisée par la résolution du conflit qui existe à l'intérieur du couple C&C.

- Fin des années 2000 : étude de formes de couplage indirect et mesure du couple C&C au niveau des paquets OO au lieu des classes.

C.2 Réponse aux questions de recherche secondaires

Étant donné que notre étude bibliographique a été répartie selon trois (3) décennies, nous allons d'abord répondre à nos questions de recherche secondaires individuellement, pour chaque décennie, avant d'émettre une réponse générale pour chaque question de recherche. Le tableau ci-dessous présente les réponses propres à chaque décennie :

Tableau C.1 Réponse aux questions de recherche secondaires par décennie

Question de recherche secondaire	Décennie	Réponse	Explications
Depuis le début des années 1970, y a-t-il une évolution ou des évolutions des concepts de cohésion et de couplage vers une plus grande précision ?	1980-1989	Oui et non	Les travaux effectués durant cette décennie nous ont convaincu de la nature subjective et ambiguë des définitions et des listes des catégories de C&C originales, qui dépendent énormément du paradigme de programmation procédurale. Ils (les travaux effectués) nous ont de même prouvé l'impact de C&C sur « les attributs de qualité logicielle ». Ceci en soit constitue une évolution notable, car nous sommes dans une position qui nous permet de toucher l'influence du couple C&C sur la qualité en général, et qui nous rend conscients des problèmes dont souffre le couple C&C – problèmes que les travaux de cette décennie ont seulement soulevés sans présenter des solutions. Il y a des propositions de métriques et de méthodes de mesurage objectives mais, comme le montreront les études subséquentes, leur rigueur est douteuse.
	1990-1999	Oui et non	Cette décennie a été marquée par l'important nombre de métriques objectives du couple C&C produites : • Métriques applicables pendant la phase codage (évaluation de la qualité). • Métriques applicables pendant la phase conception (prédiction de la qualité). • Métriques fondées sur des bases théoriques

Question de recherche secondaire	Décennie	Réponse	Explications
			solides et destinées aux applications OO. • Métriques applicables sur le système en entier (au lieu du module). Elle est aussi distinguée par la grande diversité des méthodes et techniques utilisées dans l'élaboration des métriques de C&C, comme : le tranchage, la théorie de l'information, le principe d'association de données et la détection du couplage logique à travers l'examen des rapports de changements de toutes les versions du logiciel. Mais la difficulté principale qui a été abordée concerne la création de métriques de C&C spécifiques à la COO, car toutes les métriques existantes étaient propres au paradigme de programmation procédurale et même les listes de catégorisation originales étaient fondée sur les notions et principes de ce paradigme et employaient la terminologie du langage le plus répandu à l'époque qui était FORTRAN. Malgré le grand nombre de métriques OO proposées, la définition des concepts de C&C est loin d'être précise et il n'y a pratiquement pas d'études empiriques pour valider ces métriques. De plus, nous doutons qu'ils couvrent tous les aspects et formes du couple C&C, que ce soit pour les applications OO ou pour n'importe quelle autre approche de conception.
	2000-2008	Oui	Dans cette décennie nous avons noté une évolution considérable, car : o Les chercheurs ont mieux étudié les liens entre la cohésion et le couplage et l'influence de ce lien sur la complexité et la qualité des modules. o On a aussi catégorisé les méthodes afin que le comptage des interactions entre les membres d'une classe soit un critère plus robuste. À titre d'exemple on ne traite pas les constructeurs et les destructeurs ou les *gets* et les *sets* comme des méthodes qui font du « vrai » travail fonctionnel.

Question de recherche secondaire	Décennie	Réponse	Explications
			o Une grande étude empirique a été menée dans le but de montrer la performance de certaines métriques proposées dans la décennie précédente. Nous pensons que l'approfondissement de la problématique des rapports internes au couple C&C et du couple à la qualité du logiciel a été remarquable. Comme a été un pas en avant l'introduction du concept de couplage indirect d'abord et ensuite sont étude empirique. Mais, malgré cette progression, il n'est pas difficile d'imaginer que l'on n'est pas encore au bout du chemin et que de nouvelles études critiqueront les travaux réalisées et proposerons de nouvelles définitions et de nouvelles métriques.
Est-ce qu'il existe des différences significatives dans la définition et dans l'emploi de la cohésion et du couplage ?	1980-2008	Non	La conceptualisation proposée par Yourdon et Constantine que nous avons analysée au chapitre 4 continue d'être à la base de toutes les recherches concernant les liens entre le couple C&C et la qualité du logiciel. Les concepts ont été approfondis, les définitions ont gagné en objectivité, de nouvelles mesures ont été proposées mais le « fond » conceptuel n'a pas changé.
Les métriques pour la cohésion et le couplage permettent-elles de mieux éclaircir ces concepts ?	1980-1989	Oui	Même si cette décennie n'a pas vraiment été la décennie idéale pour la production de métriques de C&C objectives, les quelques métriques proposées ont permis de mieux saisir la complexité de ces deux concepts.
	1990-1999	Oui	Le fait que certains chercheurs se sont retrouvés en désaccord concernant l'efficacité et la performance des métriques a permis d'éclaircir (par la négative) certains aspects de C&C.
	2000-2008	Oui	Même si nous sommes loin d'avoir des définitions formelles et des métriques qui satisfont les exigences des chercheurs et des praticiens, il nous semble que la catégorisation des méthodes de classes en fonction de C&C et l'approfondissement du couplage indirect ont grandement amélioré la compréhension des deux concepts.

Pour la première question de recherche secondaire, nous répondons oui et non. Oui, car nous avons noté une grande évolution vers des concepts plus clairs et moins ambigus. Et non, parce qu'avec les recherches accomplies, de nouvelles pistes sont ouvertes et beaucoup de questions sont posées. Donc ce que nous pouvons dire, c'est qu'il y a eu une évolution mais pas encore celle désirée.

En ce qui concerne la définition et l'emploi de C&C, comme nous l'avons déjà vu dans les chapitres 4, 5 et 6, les mesures de C&C ont été toujours définies comme des composants de la qualité, et employées comme des mesures de prédiction ou d'évaluation de la qualité logicielle, dans le seul et unique but d'éclaircir le couple de concepts C&C. Cet emploi est intuitivement évident car depuis toujours ils ont été associés à la modularisation, à la complexité et en général à la qualité. Donc nous répondons à la deuxième question par non, car leurs définitions et leurs utilisations sont les mêmes.

En ce qui concerne les métriques de C&C, nous avons aussi remarqué que le besoin de conception de ces dernières continue à exister malgré le grand nombre de métriques déjà élaborées. Ceci revient à la nature de ce besoin qui change continuellement avec le temps. Nous ressentons cela à travers :

- ✓ Le besoin de créer des métriques objectives.
- ✓ Le même besoin ressenti pour des métriques spécifiques à la COO.
- ✓ La nécessité de concevoir des métriques fondées sur des bases théoriques solides.
- ✓ L'utilité de métriques prédictives employées dans des phases avancées de la conception.
- ✓ Le besoin de métriques OO qui reposent sur des critères de mesurage solide.
- ✓ Le besoin de métriques OO qui considèrent les propriétés de l'entité sur laquelle elles sont appliquées.

Ce besoin changeant montre que les chercheurs se penchent plus sur le détail des différents aspects du couple C&C, donc il y a une progression dans la clarté de ce couple. Aussi les différentes disparités dans les points de vue concernant certaines métriques du C&C, et les critiques parfois présentées vis-à-vis de certaines d'entre elles, ouvrent un nouveau chemin vers la définition de nouvelles métriques n'ayant pas les défauts soulevés, constituent de même une preuve de cette progression. Convaincus par cela, nous répondons à la troisième question de recherche par un non.

C.3 Réponse à la question de recherche principale

Notre question de recherche principale était : « ***Est-ce que les concepts de cohésion et de couplage ont des définitions assez précises pour pouvoir être employés de manière efficace dans le GL ?*** »

Notre réponse est non. Les concepts C&C ne sont pas vraiment clairs et précis. Mais, malgré cela, ils sont bien employés en GL. Comme nous l'avons déjà vu, ils sont directement liés à la complexité en général et en particulier à la complexité structurelle. Donc, ils permettent d'évaluer la conception logicielle et cela en identifiant les modules mal conçus, qui peuvent éventuellement constituer une source de problèmes. Ce qui exige une restructuration de ces modules, que ce soit pendant la phase de conception ou la phase de codage.

Le grand nombre de recherches effectuées sur le couple C&C, et ayant comme objectif général l'éclaircissement de ces deux concepts, est une preuve de l'utilité et du grand emploi du couple C&C en GL. Reste à savoir si cette utilisation est efficace ou non. Nous pensons qu'elle ne l'est pas vraiment en raison de l'ambiguïté qui existe encore autour de ces deux concepts. Rappelons que leurs définitions actuelles sont pratiquement les mêmes que celles données dans les années 1970 par Yourdon et Constantine. Le remède envisagé durant ces trois (3) décennies, était la définition de métriques objectives. Mais, ce nombre de métriques n'a cessé d'augmenter avec, à chaque fois, une critique des métriques existantes. Donc, nous

pensons qu'il est encore fort difficile d'adopter des métriques normalisées et objectives capables de mesurer les concepts C&C.

C.4 Apprentissage personnel

En entamant cette recherche, j'ai[43] pu saisir la vraie difficulté de la mesure de la qualité et cela en essayant de suivre le trajet de la recherche de deux minuscules composants de quelques uns de ses multiples attributs et à travers lesquels j'ai ressenti les embarras que les chercheurs traitant ce sujet assez délicat ont rencontré. En outre, j'ai pu saisir les problèmes des concepts du GL, qui restent quand même flous en dépit du grand nombre de normes par lesquelles on a essayé de les normaliser et des métriques proposées pour les quantifier et enfin les clarifier.

Par ailleurs, grâce au travail sur la littérature, j'ai amélioré mon esprit de critique et d'analyse et cela en travaillant sur un grand nombre d'articles et de livres qui, même s'ils traitent le même sujet et poursuivent le même objectif, présentent des visions différentes, parfois complémentaires et parfois opposées.

Ce travail m'a permis d'acquérir de nouvelles connaissances dans des sous-domaines particuliers de la qualité logicielle, parmi lesquels :

- La complexité, avec tous ses différents types.
- La modularisation.
- Les différents modèles de qualité, leurs points de similitude et de divergence.
- Les différentes définitions du couple C&C qui au fond ne sont pas vraiment différentes car toutes découlent de la même source.
- Les différentes et multiples catégorisations du couple C&C.
- Les métriques de complexité et du couple C&C produites.

---

[43] L'emploi du pronom personnel « je » rends le contenu de cette section personnel.

Ce travail m'a aussi permis de maîtriser les méthodes et les techniques sur lesquelles les chercheurs se sont fondés pour concevoir des métriques objectives de C&C, telles que : le tranchage, le principe d'association, la théorie de l'information, la technique de détection de couplage logique à travers les versions d'un logiciel et la technique de détection de certaines formes de couplage indirect.

C.5 Difficultés rencontrées

Étant donné que notre projet de recherche, était basé sur une analyse approfondie de la littérature, les difficultés que nous avons rencontrées au niveau de la littérature sont :

- Le non accès à quelques articles que nous aurions aimé consulter.
- La terminologie du domaine GL qui est assez variable (pas de normes adoptées) que ce soit entre les articles étudiés, ou les modèles analysés, ou même les normes consultées.
- La non clarté des idées présentées par certains articles étudiés et aussi le manque de preuves de certaines affirmations.
- Le manque de précision de certains termes employés à plusieurs.

En ce qui concerne la démarche que nous avons adoptée pour mener notre recherche, nous avons rencontré une seule difficulté qui était le choix de cette démarche. Comme nous l'avons déjà mentionné dans le premier chapitre, nous avons été face à plusieurs démarches en ignorant complètement la plus avantageuse car cette recherche est unique et les questions de recherche sont (à notre connaissance) abordées pour la première fois.

C.6 Limites et contraintes

Ce projet de recherche présente deux (2) principales limites :

- La première est que ce travail a une forte composante subjective, que ce soit pour le choix de la matière étudiée (livres, articles et normes) ou pour les analyses et les critiques effectuées, et sur lesquelles nous nous sommes principalement reposés pour répondre à nos questions de recherche. En effet,

ces analyses et ces critiques sont subjectives en raison de leur caractère personnel et du fait qu'elles présentent notre point de vue et notre propre vision des choses.

- La deuxième est que le nombre d'articles choisis dans notre étude est petit par rapport au grand nombre de publications traitant du couple C&C parues durant les trois (3) décennies considérées. Mais, nous jugeons qu'il s'agit d'un nombre raisonnable pour un travail de recherche au niveau de la maîtrise.

C.7 Contributions

Les principales contributions de cette recherche sont :

Apporter quelques éléments pour approfondir la réflexion sur la cohésion et le couplage. Éléments qui, même s'ils n'ont pas la prétention de constituer une base très solide, sont, à notre avis, un point de départ pour appréhender la situation antérieure et actuelle du couple C&C.

Percevoir la difficulté d'uniformiser la transformation de l'idée abstraite qu'enveloppe le couple C&C en mesures objectives aidant à éclaircir cette idée même, dans le but de contenir la qualité logicielle. Autrement dit, cette étude a essayé de confirmer l'affirmation de Tom deMarco mais inversée : « *vous ne pouvez pas mesurer ce que vous ne maîtrisez pas* » (Maffezzini, Premiana et Ventimiglia, 2004).

C.8 Travaux futurs

Les limites de notre étude (présentées plus haut) sont en même temps des ouvertures pour des travaux futurs :

- Même travail mais basé sur une matière et une démarche différentes aux nôtres, pour voir si c'est la seule progression qu'a subi le couple C&C ou s'il y en a bien d'autres ;

- Enquête auprès des théoriciens et des chercheurs avec des analyses statistiques pour rendre moins subjectives certaines de nos considérations.

APPENDICE A

TRANSCRIPTIONS/RÉSUMÉS DES ARTICLES ANALYSÉS

Dans cet appendice nous présentons le travail « brut » que nous avons réalisé sur les articles avant d'en faire les synthèses présentées dans les chapitres précédents. Il ne s'agit pas de vrais résumés mais d'un mélange de traductions[44], de transcriptions et de résumés. Pour chaque article nous avons approfondi surtout les parties les plus difficiles[45], ce qui crée parfois des déséquilibres dans la structure du texte (et donc des difficultés de lecture). Même s'il s'agit de textes « bruts », il nous a semblé important de les insérer dans notre étude, d'une part parce qu'il est probable que d'autres personnes avec une formation en génie logiciel comme la nôtre ont les mêmes genres de difficultés et de l'autre parce que ces textes sont un pont assez solide entre les synthèses et les originaux.

A.1 Couplage et cohésion : Années 1980-1989

Dans cette section, nous résumons quatre (4) articles traitant le couple C&C publiés dans les années 1980. Ces articles sont :

1. « Une métrique discriminante de la cohésion »
2. « Critères de modularisation du logiciel »
3. « Analyse de la cohésion et du couplage pour un système ayant tendance à avoir des erreurs »
4. « La relation entre les tranches et la cohésion du module »

---

[44] Traduction parfois assez difficile.
[45] Pas difficiles en absolu mais en ce qui concernaient nos connaissances.

A.1.1 Emerson (1984) « Une métrique discriminante de la cohésion »

La « modularisation » est une méthode de conception de logiciel adoptée, non seulement pour rendre le problème à résoudre moins complexe, mais aussi pour favoriser les changements requis dans les différentes phases du CVL.

Parmi les critères qui servent utilement à évaluer l'efficacité de la technique utilisée dans l'adoption de cette méthode, il y a la cohésion qui est une « *métrique prédictive et subjective* », parce que sa détermination requiert « *l'interprétation des caractéristiques du module dans un cadre de définitions informelles* » (Basili, 1977).

A.1.1.1 Conception de logiciel et propriétés du module

La cohésion a été définie comme un attribut des modules par Constantine —et aussi bien par Myers mais sous le nom de « force de module », qui indique « *le degré d'union fonctionnelle du module* » (Basili, 1977). Elle est également définie dans la norme IEEE 729-1983, comme : « *le degré fonctionnel de liaison des tâches exécutées par un simple module d'un programme* » (IEEE 729-1983, 1983).

Yourdon et Constantine ont, par la suite, défini les sept (7) célèbres niveaux de cohésion propres à des principes d'association différents. En pratique, on n'a pas souvent recours à ces niveaux. D'ailleurs, dans son étude, Emerson (1984) les réduit à trois types :

- Type I : fonctionnelle, séquentielle et communicationnelle. Les modules de ce type de cohésion effectuent des actions contribuant à l'accomplissement de la même fonction.
- Type II : procédurale, temporelle. Les actions des modules de ce type sont liées par leurs ordres d'exécution.
- Type III : logique, fortuite. Un module qui comprend des actions complètement indépendantes les unes des autres est dit de type III de cohésion.

A.1.1.2 Introduction à la théorie des graphes

La métrique définie dans cet article est « *basée sur une propriété de la théorie des graphes qu'on emploie pour mesurer la proximité de l'interaction entre les chemins des flux de contrôle et les références aux variables* ».

« *Une nouvelle propriété de sous-ensemble de sommets, appelée aussi « cohésion » est introduite* ». Avant de définir la cohésion d'un sous-ensemble de sommets, l'auteur présente quelques définitions, terminologies et notations basées sur l'étude de Biggs et Norman (1974), et qui seront utilisées dans son article. Dans ce qui suit, nous allons présenter ces dernières.

Un graphe de flux F est défini comme : « *un graphe dirigé avec deux sommets I et T, tel que :*

Il n y a pas d'arc qui vient de T,

Pour chaque x □ VF, il y a un chemin de I à T et qui passe par x ».

VF est l'ensemble de sommets du graphe F et EF est son ensemble d'arêtes.

Un chemin complet dans F est « *un chemin de I vers T. L'ensemble des chemins dans F est noté IIF, si A □ VF, alors IIA est l'ensemble des chemins complets qui passent au moins par un élément de A* ».

L'auteur définit ensuite la cohésion d'un sous-ensemble de sommets :

Si A □ VF, la cohésion de A avec le respect de F est :

$k(A,F)$ [46]$= (\dim A)\,|A| / (\dim F)\,|F|$

Où :

---

[46] Toujours $K(A,F) \leq 1$, dans le cas où : A=VF, $k(A,F) = 1$.

- Dim (A) : est le nombre d'éléments dans le plus grand ensemble linéaire et indépendant des chemins complets passant par A.
- |A| : est le nombre d'éléments dans A.
- Idem pour dim (F) et |F|.

En d'autres termes, la cohésion d'un sous-ensemble de sommets A est le rapport du nombre d'éléments de ce sous-ensemble multiplié par le nombre d'éléments du plus grand ensemble des chemins complets passant par A, et le nombre d'éléments de tout l'ensemble F, multiplié aussi par le plus grand ensemble des chemins complets passant par F.

Le calcul de la métrique de cohésion, nécessite le calcul de la dimension d'un ensemble de sommets du graphe de flux d'un programme. Donc, l'auteur présente le calcul de cette entité, sous forme d'un théorème :

Si F est un graphe de flux et A □ F, alors, dim A = |RE(A)| - |RV(A)| + 2

Où :

- RV(A) : est l'ensemble de sommets x dans VF, tel qu'il y a un chemin complet dans IIA qui passe par x.
- RE(A) : est l'ensemble des arêtes a dans EF tel qu'il y a un chemin complet dans IIA qui passe par a.

Dans son étude, Emerson (1984) démontre la justesse de ce théorème par des preuves mathématiques, que nous ne présentons pas. Dans ce qui suit, nous allons nous intéresser à la métrique de cohésion qu'il propose (section A.1.1.3), ainsi qu'à ses propriétés (section A.1.1.4).

A.1.1.3 La métrique de cohésion

Le module (procédure, fonction, sous-programme) auquel la métrique d'Emerson (1984) peut être appliquée, doit vérifier les conditions suivantes :

« *Il a un seul point d'entrée,*

Si une instruction exécutable ne contient pas une référence à une variable, ou elle termine le module (par exemple, RETURN) ou elle a une seule instruction qui la suit dans le graphe de contrôle ».

Les variables contenues dans ce module sont définies comme « *des identificateurs qui représentent des valeurs changeables avec le temps, c.-à-d. les littéraux*[47] *et les constantes symboliques sont exclues* ».

Une fois qu'on est assuré que le module vérifie ces conditions, on construit ce qu'on appelle un graphe de flux réduit, qui exige le passage par deux étapes :

1. Construire le graphe de flux F' de M, dont chaque sommet représente une variable exécutable dans M, et dont les arêtes relient les instructions qui se suivent dans l'ordre d'exécution. On rajoute à ce graphe, le sommet T, auquel tous les nœuds des instructions return, stop, etc. sont liés. Le sommet I y est déjà, et il correspond au nœud de la première instruction exécutable.

2. Construire le graphe de flux réduit F de M, à partir de F', et cela en supprimant tous les sommets des instructions exécutables qui ne contiennent pas une référence à une variable. Les arêtes qui se terminaient dans un sommet x qui appartient à F', qui est supprimé de F, se terminent maintenant dans son successeur[48].

Ensuite, on définit l'ensemble des références, défini pour une variable i du graphe F, noté Ri, comme : « *l'ensemble de sommets de F correspondants aux instructions exécutables qui réfèrent la* $i^{ème}$ *variable*».

$$\bigcup_{i=1}^{v} R_i = VF - \{T\}.$$ [49]

Où :

---

[47] Libellés, des unités lexicales qui représentent explicitement des valeurs.
[48] La deuxième condition des modules considérés par l'auteur assure que chaque sommet a un et un seul successeur.
[49] Le sommet T est enlevé car il ne correspond à aucune instruction exécutable.

v : est le nombre de variables du module M.

Enfin, on calcule la cohésion du module qui est « *la moyenne des ensembles de références* ». La formule de calcul est la suivante :

$K(F) = \sum k (Ri, F) / v$ (i allant de 1 à v)

où : $K(Ri, F) = (|Ri| * \dim Ri) / (|VF - \{T\}| * \dim F)$

A.1.1.4 Propriétés de la métrique

La métrique K a une limite inférieure, ce que l'auteur prouve en démontrant le théorème suivant :

Théorème : Si F est le graphe de flux d'un programme avec les ensembles des références des variables R1,R2,…,Rv, s le nombre d'instructions exécutables qui référencient une variable (c.-à-d., $s = |VF - \{T\}|$), d la dimension, et k la métrique de cohésion, alors :

$K \geq \max \{1/vs, 1/vd\}$

Avec :

- vs : le nombre de variables du module multiplié par le nombre d'instructions exécutables qui référencient une variable du module, et
- vd : le nombre de variables du module multiplié par la dimension du graphe de flux du module.

A.1.1.4.1 Cohésion de graphe de flux de type II

Les éléments de traitement d'un module de cohésion de type II, sont liés par le fait qu'ils « *peuvent être exécutés dans un flux séquentiel de contrôle* ».

La construction du graphe de flux de ce genre de module est définie comme suit :

« *Si F1,...,Fn sont des graphes de flux avec les sommets initiaux Ii,...,In et des sommets terminaux T1,...,Tn, alors nous définissons F1F2...Fn comme le graphe de flux construit en identifiant Ti avec Ii+ 1 pour i= 1,2,...,n-1* ».

Théorème : On considère n copies de F ($n \geq 2$) (nommés F1F2...Fn), chaque copie Fi a des ensembles de références Ri1,...Riv, avec Rij est la copie de Ri dans VF. On combine F1,...,Fn, dans le même graphe de flux F1F2...Fn, en considérant les Rij les sous-ensembles des nœuds du nouveau graphe.

$(d/n)*K(F) \leq K(F1F2...Fn) \leq (1/n2)*K(F)$

A.1.1.4.2 Cohésion de graphe de flux de type III

Les éléments de traitement de ce type de module n'ont aucune relation les uns avec les autres. La construction du graphe de flux de ce genre de module s'établit sur la définition suivante.

Définition : « *Si F1,...,Fn sont des graphes de flux avec les sommets initiaux Ii...,In et des sommets terminaux T1,...,Tn alors nous définissions le graphe de flux F1\|F2\|...\|Fn construit en identifiant les sommets terminaux T1,T2,...,Tn par le sommet T, et en ajoutant un nœud I, et une arête allant de I vers chaque sommet I1,I2,...,In* ».

L'auteur démontre ensuite le théorème suivant :

Théorème : $(2/n^2) * k(F) \geq k(F1|F2|...|Fn)$

Après la démonstration du théorème, l'auteur présente des considérations pratiques.

Normalement, un module de niveau I de cohésion devrait rapprocher la valeur (q/c)/s.

- q est supposé être le nombre moyen des références de variables par instruction exécutable (la valeur de q dépend du langage de programmation), c'est une constante de proportionnalité entre le nombre de variables du module et le nombre d'instructions exécutables.

– s représente le nombre d'instructions exécutables qui référencient une variable, on note qu'un module aurait en moyenne (sq)/(cs) nœuds dans chaque ensemble de références.

Et à partir des théorèmes présentés dans la section précédente, « *à mesure que le nombre d'instructions augmente, la cohésion diminue légèrement plus rapidement de 1/s pour les modules de type II de cohésion, et de* $1/s^2$ *pour les modules de type III de cohésion* ».

Ces hypothèses ont été vérifiées par une étude empirique menée par l'auteur sur des modules tirés des deux premiers chapitres du livre « *Software Tools* » de Kernighan et Plauger (1976). Les résultats obtenus ont montré que tous ces modules ont une cohésion de type I ; ce qui est normal, puisque le but de ce livre est d'enseigner la conception du logiciel en se basant sur des exemples de programmes bien conçus.

A.1.1.5 Références

Dans son article, l'auteur utilise quatorze (14) références :

- Quatre livres dont un est celui de Yourdon et Constantine (1979).
- Sept articles, dont :
  - Trois (3) qui datent des années 1970, et
  - Quatre (4) des années 1980.
- Une norme, il s'agit d'IEEE 729-1983 glossaire de la terminologie du génie logiciel.

A.1.2 Card, Page et McGarry (1985) « Critères de modularisation du logiciel »

Dans l'introduction de cet article, les auteurs présentent les deux éléments principaux qui les ont poussés à entreprendre leur recherche :

- « *Un observateur indépendant du processus de développement peut difficilement déterminer le niveau de cohésion et de couplage [...] d'un module* ».
- « [...] *Il n'y a ni base théorique, ni évidence empirique pour employer la taille des modules comme critère de modularisation* ».

Il existe trois critères de modularisation majeurs qui sont : la cohésion, le couplage et le masquage de l'information. Même si le couplage est aussi important que la cohésion, les auteurs ne le considèrent pas dans leur étude, car il implique que « *l'on considère plus d'un module en même temps* ». Donc, ils étudient la cohésion et la taille, même s'il n'a jamais été prouvé que cette dernière constitue un critère de modularisation.

Dans ce qui suit, nous allons nous intéresser aux données analysées (section A.1.2.1), aux résultats de l'analyse (section A.1.2.2) et aux références employées par Card, Page et McGarry (1985) dans leur étude (section A.1.2.3).

A.1.2.1 Données analysées

Dans leur recherche empirique, les auteurs ont analysé cinq (5) grands projets de développement de logiciel programmés en FORTRAN. Aux projets ont participé 26 programmeurs/analystes qui ont créé un total de 453 modules.

Avant d'ébaucher l'étude proprement dite, ils établissent les définitions suivantes :

- Module : « *sous-programme FORTRAN, ou la plus petite unité d'un programme indépendamment compilable* ».
- Coût de module : « *nombre d'heures par instruction exécutable* ».
- Taux d'erreur : « *nombre d'erreurs par instruction exécutable* ».

À cause des difficultés d'application des mesures de cohésion présentées dans la littérature, pour « mesurer » la cohésion, ils ont utilisé des listes de vérification habituellement employées par les programmeurs pour cette fin. À l'aide des listes de vérification, les 453 modules ont été divisés selon leurs degrés de cohésions dans les trois groupes présentés dans le tableau ci-dessous.

Tableau A.1 Distribution des modules selon leur cohésion (Source : Card, Page et McGarry, 1985)

MODULE STRENGTH	NUMBER OF FORTRAN MODULES	MEAN EXECUTABLE STATEMENTS	MEAN DECISIONS PER EXECUTABLE STATEMENT
LOW	90	77	0.29
MEDIUM	176	60	0.32
HIGH	187	48	0.32

Et divisés aussi en trois groupes selon leurs dimensions, le tableau ci-dessous présente cette distribution.

Tableau A.2 Distribution des modules selon leur dimension (Source : Card, Page et McGarry, 1985)

MODULE SIZE	NUMBER OF FORTRAN MODULES	EXECUTABLE STATEMENTS	MEAN DECISIONS PER EXECUTABLE STATEMENT
SMALL	154	1 TO 31	0.31
MEDIUM	148	32 TO 64	0.31
LARGE	151	65 OR MORE	0.32

A.1.2.2 Résultats de l'analyse

En partant des distributions des deux tableaux précédents, ils ont étudié la corrélation entre la cohésion, le taux d'erreurs et le coût de développement d'une part, et d'autre part, la taille du module, le taux d'erreurs et le coût de développement. Le tableau ci-dessous montre qu'il y a une corrélation négative significative entre la cohésion d'un module et son taux d'erreurs (-0.35), et entre la taille d'un module et son coût de développement (-0.31).

Tableau A.3 *Contingency Results* (Source : Card, Page et McGarry, 1985)

CRITERIA	EFFECT CONTROLLED	CORRELATIONS[a]		LINE
		FAULT RATE	COST RATE	
MODULE STRENGTH	NONE	−0.35[b]	−0.19	1
	SIZE	−0.32[b]	−0.27[b]	2
	PROGRAMMER	−0.21	0.10	3
MODULE SIZE	NONE	0.20	−0.31[b]	4
	STRENGTH	0.19	−0.38[b]	5
	PROGRAMMER	0.27[b]	−0.41[b]	6

Le tableau ci-dessus montre que « *les modules ayant une forte cohésion (normalement petits) ont tendance à avoir un coût faible tout comme les gros modules (indépendamment de leur cohésion)* » (Card, Page et McGarry, 1985).

Les deux figures ci-dessous montrent clairement que :

- Plus la cohésion est forte, plus le nombre d'erreurs est faible (50 % des modules de forte cohésion contiennent zéro erreur),
- Plus la taille ou la dimension du module est grande, plus son coût est faible (46 % des grands modules sont de coût faible).

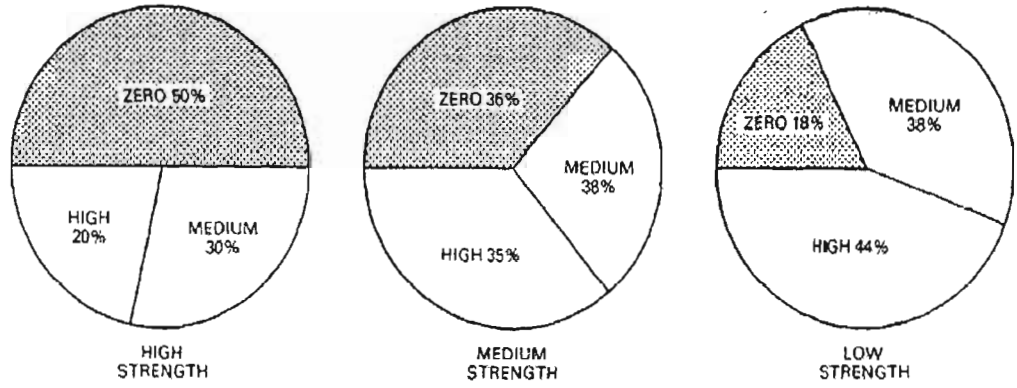


Figure A.1 Taux d'erreurs par classe de cohésion des modules (Source : Card, Page et McGarry, 1985)

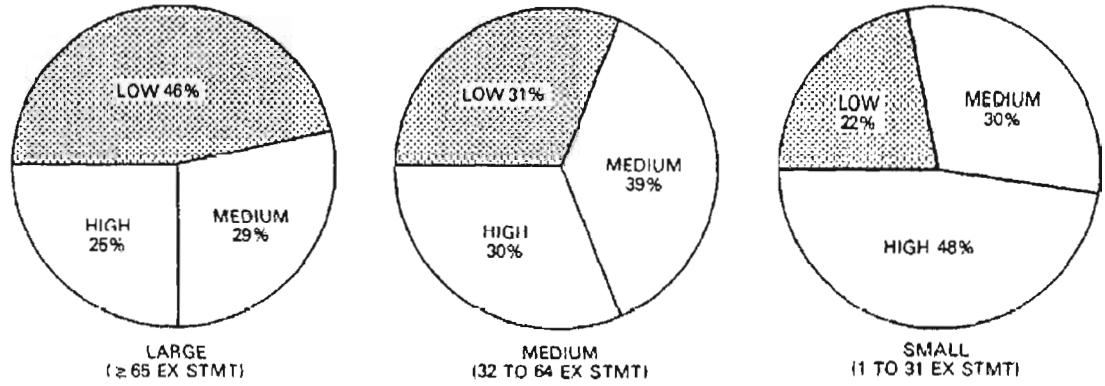


Figure A.2 Coût de développement par classe de dimension de modules (Source : Card, Page et McGarry, 1985)

Les auteurs ont aussi remarqué qu'en contrôlant la dimension du module, la corrélation entre la cohésion du module et son coût augmente, et qu'en contrôlant la cohésion du module, la corrélation entre la dimension du module et son coût augmente aussi. Ceci les a mené à induire qu' « *en général, les modules de haute cohésion (habituellement petits) tendent à être moins coûteux, et aussi les grands modules tendent à être moins coûteux (indépendamment de leur degré de cohésion)* ».

Enfin, les auteurs attirent l'attention du lecteur sur un critère de modularisation souvent négligé qui est la performance du programmeur, du fait que « *les modules de forte cohésion sont associés aux programmeurs qui produisent des modules de coût faible et ayant un taux d'erreurs réduit* ». Ici, les auteurs ne parlent pas de dimension du module car ils ont remarqué que cette dernière ne veut rien dire pour un bon programmeur.

Finalement, ils présentent un conseil qu'ils croient pouvoir tirer de leur analyse : « *Généralement, les programmeurs devraient être encouragés à écrire des modules de forte cohésion et à rendre ces modules assez grands pour englober une fonction entière. Puisque les modules de faible cohésion sont susceptibles d'être plus grands que la moyenne, la taille du module peut avoir un effet favorable indirect sur le taux d'erreurs. Cependant, les avantages de coût associés à de grands modules permettent de dire que les grands modules de forte cohésion doivent également être acceptables. Les grands modules peuvent être appropriés pour certains types de logiciel (par exemple, algorithmes mathématiques)* ».

A.1.2.3 Références

Les auteurs ont utilisé quatorze références :

- Un livre, « *Structured Design* », de Myers, Stevens et Constantine publié en 1974.
- Dix articles, dont trois publiés dans les années soixante-dix et sept dans les années 1980.
- Une communication des années 1970.
- Deux références dont nous ignorons la nature.

A.1.3 Selbi et Basili (1988) « Analyse de la cohésion et du couplage pour un système ayant tendance à avoir des erreurs »

Dans cet article, les auteurs commencent par donner une définition qu'ils appellent « intuitive » de la cohésion et du couplage. La cohésion étant « *le volume d'interactions* ***à l'intérieur*** *des composants (ex. sous-système, module) du système logiciel* », et le couplage

étant « *Le volume d'interactions* ***entre*** *ces composants* ». La cohésion et le couplage sont mesurés par les « *interactions intra système en termes d'association de données* ». Les erreurs ont été « *collectées à partir de l'accomplissement de la conception de haut niveau jusqu'au test du système, à noter que quelques-unes de ces erreurs surviennent des opérations du système* ».

Dans ce qui suit nous allons parler de sous-programme (routine) et sous-système. Nous présentons alors les définitions de ces termes comme données par les auteurs :

- ***Routine :*** Il s'agit d'un programme principal, d'une procédure ou d'une fonction.
- ***Sous-système :*** C'est un large ensemble de sous-programmes liés entre eux pour former une entité de système exécutable.

Cette étude a été réalisée sur la nouvelle version d'un projet qui avait au début 100 000 lignes de codes et à la fin 140 000 lignes. Le produit final est constitué de 77 sous-systèmes pour un total de 163 fichiers sources, qui contiennent en moyenne 2,8 routines. Le projet a duré 16 mois avec un maximum de 23 personnes. Dans leur recherche, les auteurs ont suivi le paradigme « *goal-question-metric* » (Basili et Weiss, 1984) fondé sur ces trois étapes :

1. Définir les buts de la collecte et de l'analyse des données.
2. Raffiner les buts pour déterminer la liste des questions spécifiques.
3. Établir des métriques et des catégories de données appropriées.

Afin de décrire le processus d'association de données adopté dans cette étude, les auteurs trouvent nécessaire de définir les trois premiers niveaux d'association de données suivants :

- Association de données potentielle : est un triplet ordonné (p, x, q), où p et q sont des procédures et x est une variable à l'intérieur de la portée statique de ces deux dernières.
- Association de données utilisée : est une association de données potentielle, où p et q utilisent la variable x pour une référence ou une affectation.

- Association de données réelle : est définie comme une association de données dans laquelle p affecte une valeur à x et q référencie x.

Il existe bien d'autres niveaux plus élevés d'association de données. Mais, dans cette étude, les auteurs se contentent de l'association de données réelle qui est quand même assez difficile à calculer puisqu'elle nécessite beaucoup de mémoire dans le stockage de l'information qui en résulte. Mais, d'autre part, elle n'exige pas une analyse complexe des flux de données et fournit une mesure efficace de l'association de données.

L'application de la technique statistique de groupage (*Clustering*) sur l'information obtenue du calcul de l'association de données réelle du système, a permis aux auteurs de faire ressortir la description hiérarchique du logiciel. Pour ce faire cinq (5) outils ont été développés, à savoir :

- *Source_bind* : Lit le code source et produit un fichier qui contient l'information concernant l'utilisation des variables, des procédures et des fonctions.
- *Link_bind* : Prend les sorties d'une ou de plusieurs exécutions de *source_bind* et combine ces informations comme fait un éditeur, en donnant à chaque objet de donnée un nom unique.
- *Matrix_bind* : Prend la sortie de *link_bind* et construit une matrice qui contient une ligne et une colonne de chaque procédure ou fonction dans le code source.
- *Fold_bind* : Lit la sortie de *matrix_bind* et crée une matrice de dissimilitude.
- *Cluster_program* : Lit la sortie de *fold_bind* et produit une description du système sous forme d'arbre.

Tel que déjà énoncé, cet ensemble d'outils est utilisé dans la production des descriptions hiérarchiques des 77 sous-systèmes étudiés. « *Ces descriptions sont présentées sous forme d'arbres connectés, dont les sous-arbres sont des regroupements de sous-programmes qui forment des groupages naturels fondés sur les critères d'association de données* ». Et, chacun de ces groupages contient soit des sous-programmes ou des sous-

arbres. « *À chaque groupage d'un sous-système est associé un nombre allant de 0 à 100* » (Selbi et Basili, 1988). Ce nombre désigne la nature de l'association entre les sous-programmes du même groupage, et est interprété par le ratio suivant : le couplage du groupage avec d'autres groupages dans le sous-système / la force interne du groupage

La figure ci-dessous présente les formulaires employés pour la collecte de données. Elle est faite à partir des :

- Inspections.
- Feuilles de travail.
- Rapports de dysfonctionnement du système.

Dans cet article les auteurs, emploient le terme « erreur » pour désigner les fautes commises par le développeur et qui engendrent des fautes (*faults*) et des pannes (*failures*). La figure ci-dessous met en évidence la différence entre *fault* (la manifestation dans un document d'une erreur des programmeurs), qui est détectée durant les inspections de conception et de codage, et *failure* (panne lors de l'exécution du programme), qui est décelée par les sommaires de feuilles de travail, et les rapports de dysfonctionnement.

Development phase	Data collection form	Data recorded	
		Faults	Failures
High-level design	Design inspection	X	
Low-level design	Design inspection	X	
Coding and unit test	Error summary worksheet (ESW)	X[1]	X
After unit test completion	Engineering inspection	X	
Integration test	Error summary worksheet (ESW)		X
System test	System trouble report (STR)		X
Field test and operation	Trouble report (TR)		X

Figure A.3 Les formes de collections des données utilisées dans les phases de développement

Les auteurs présentent neuf (9) tableaux montrant les erreurs trouvées dans les différentes phases. À titre d'exemple la figure ci-dessous présente les erreurs détectées lors des inspections. Les erreurs sont organisées en deux catégories : majeure et mineure.

Severity	Inspection type		
	Design	Engineering	All
Major	275	158	433
Minor	175	162	337
All	450	320	770

Figure A.4 Distribution des erreurs par degré de sévérité et par type d'inspection

La figure ci-dessous présente le nombre d'erreurs et l'effort requis pour la correction de ces erreurs dans les sous-programmes de haut et de faible degré de couplage/cohésion.

On note que pour les auteurs, les sous-programmes qui ont un rapport couplage/cohésion ayant une valeur supérieure à 86 sont dits de haut niveau (désignée par le terme « *High* » dans la figure ci-dessous) et ceux ayant une valeur inférieure à 86 sont dits de bas niveau (désignée par le terme « *Low* » dans la figure ci-dessous).

Subsystem coupling/ strength	Errors				Error correction hours			
	per KLOC		Total		per KLOC		Total	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
High	1.54	3.95	0.44	0.99	2.80	7.53	0.88	2.69
Low	0.31	1.16	0.15	0.52	0.91	4.51	0.42	2.39
Overall	1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

Figure A.5 [50]Distribution des erreurs et l'effort de correction des erreurs selon le rapport couplage/cohésion[51] du sous-système

Les sous-systèmes avec un rapport de couplage/cohésion élevé comportent plus d'erreurs (que ce soit par 1000 lignes de code ou en considérant tout le code) que ceux ayant un rapport faible. On remarque aussi que l'effort requis pour la correction des erreurs contenues dans un sous-système à couplage fort est très important, comparé à celui requis pour un de faible couplage.

Les auteurs catégorisent les sous-programmes (routines) en quatre catégories selon leur ratio couplage/cohésion :

- *« 4_Highest : le quartile le plus élevé des rapports de couplage/cohésion pour les faisceaux dans un sous-ensemble.*
- *3_Higher : le prochain quartile inférieur des rapports de couplage/cohésion pour les faisceaux dans un sous-ensemble.*
- *2_Lower : le prochain quartile inférieur des rapports de couplage/cohésion pour les faisceaux dans un sous-ensemble.*
- *1_Lowest : le prochain quartile inférieur des rapports de couplage/cohésion pour les faisceaux dans un sous-ensemble.* »

---

[50] Dans ce tableau et dans ceux qui le suivent, on présente les moyennes et les écarts type du nombre d'erreurs par 1000 lignes de code (KLOC), le nombre d'erreurs total, l'effort de correction d'erreurs par KLOC, et l'effort de correction d'erreurs total.

[51] Pour les auteurs *strength,* traduit par force, est synonyme de cohésion.

Routine coupling/ strength	Errors				Error correction hours			
	per KLOC		Total		per KLOC		Total	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
4_Highest	2.27	4.58	0.59	1.04	5.86	10.98	1.94	4.20
3_Higher	1.15	3.13	0.34	0.74	2.19	6.84	0.72	2.54
2_Lower	1.45	4.19	0.44	1.18	1.57	4.27	0.49	1.61
1_Lowest	0.28	1.11	0.15	0.49	0.21	1.09	0.06	0.29
Overall	1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

Figure A.6 Distribution des erreurs et de l'effort de correction des erreurs selon le rapport couplage/cohésion de procédure

La figure ci-dessus montre que « *les sous-programmes avec le plus bas rapport de couplage/cohésion contiennent 8.1 fois moins d'erreurs par 1000 lignes de code que les autres procédures, et les erreurs sont 27.9 fois moins coûteuses à corriger* ».

A.1.3.1 Références

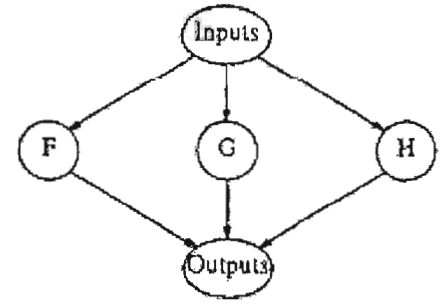
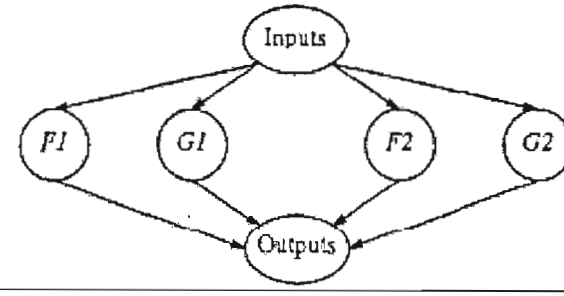
Dans cet article, les auteurs ont utilisé vingt-quatre (24) références, dont :

- Cinq (5) livres
- Treize articles, dont :
    - Trois (3) des années 1970, et
    - Dix (10) des années 1980
- Cinq (5) rapports techniques
- Une (1) communication

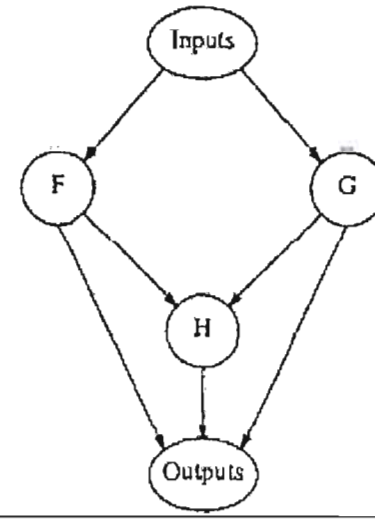
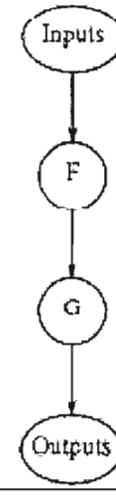
A.1.4 Ott et Thuss (1989) « La relation entre les tranches et la cohésion du module »

Ott et Thuss (1989) regroupent les sept (7) niveaux de cohésion de Yourdon et Constatine (1979) en quatre (4) niveaux, comme le montre le tableau ci-dessous.

Tableau A.4 Les quatre (4) niveaux de cohésion illustrés avec les graphes de flux d'éléments de traitement correspondats à chaque niveau

Niveau de cohésion	Définition	Exemple de graphe de flux d'éléments de traitements[52]
Cohésion faible	Les modules effectuant au moins deux traitements de données distincts sont dits de cohésion faible. Cette classe de cohésion regroupe les cohésions accidentelle et temporelle de Yourdon et Constantine (1979). La cohésion temporelle ne peut pas être différenciée de la cohésion accidentelle car les relations temporelles ne peuvent pas être analysées en se basant seulement sur des graphes de flux.	
Cohésion de type contrôle	Un module a une cohésion de type contrôle quand « *le seul rapport des éléments de traitement est le fait de faire partie de structures de contrôle commue* ». Cette classe de cohésion regroupe les cohésions logique et procédurale de Yourdon et Constantine (1979).	

[52] Ces graphes sont tirés de l'article.

Niveau de cohésion	Définition	Exemple de graphe de flux d'éléments de traitements[52]
Cohésion de données	Cette classe est équivalente à la classe communicationnelle de Yourdon et Constantine (1979), et est caractérisée par le fait que ses éléments de traitement partagent des données.	
Cohésion forte	Cette classe regroupe les cohésions séquentielle et fonctionnelle. Un module de cette classe de cohésion est soit associé à un graphe de flux d'éléments de traitement séquentiel ou englobe un simple traitement de données calculant une seule sortie.	

Leur étude étant fondée sur la relation entre les tranches et la cohésion du module, les auteurs présentent leur méthode de découpage du module en tranches. Dans ce qui suit, nous allons présenter cette méthode (section A.4.1) à travers des exemples et citer les références utilisées par les auteurs (section A.4.2) pour mener leur recherche.

A.1.4.1 Méthode de découpage du module en tranches

Les auteurs commencent par présenter quelques définitions des éléments de la méthode de découpage du module en tranches, commençant par définir un critère de découpage en tranches, comme « *un tuple <i, V>, où i dénote un nombre spécifique d'instructions dans un module M et V est un sous-ensemble des variables de M* ».

Considérons un simple exemple (Figure A.7[53]).

```
 1  procedure SumAndProduct( N : integer;
 2                           var SumN  : integer;
 3                           var ProdN : integer );
 4  var
 5     I : integer;
 6  begin
 7     SumN := 0;
 8     ProdN := 1;
 9     for I := 1 to N do begin
10        SumN := SumN + I;
11        ProdN := ProdN * I
12     end
13  end;
```

Figure A.7 Un module qui calcule la somme et le produit des premiers N entieds (Source : Ott et Thuss, 1989)

Le module de la figure ci-dessous représente une tranche du module de la Figure A.7, avec comme critère de découpage en tranches : C= < 13, {SumN} >.

[53] Les exemples (les figures de A.1 à A.13) sont tirés de l'article

```
 1  procedure SumAndProduct( N : integer;
 2                           var SumN  : integer;
 3                           var ProdN : integer );
 4  var
 5     I : integer;
 6  begin
 7     SumN := 0;
 9     for I := 1 to N do begin
10        SumN := SumN + I;
12     end
13  end;
```

Figure A.8 Tranche du module SumAndProduct obtenue avec comme critère de découpage en tranches C= <13, {SumN}>

Ensuite, les auteurs définissent un profil de tranche (*slice profile*), comme « *une représentation pratique pour mettre en évidence des patrons de tranche dans un module* ». Comme exemple, le profil de tranche pour le module *SumAndProduct* est donné par la figure ci-dessous, « *La colonne "ligne" contient le numéro de ligne de chaque instruction du module. Chaque colonne avec comme nom le nom de la variable dans le profil de tranche correspond à la tranche associée à cette variable. Toutes les rangées dans le profil identifié par une barre verticale "\|" sont des instructions incluses dans une tranche pour une variable particulière, autrement la rangée est vide* ».

Line	SumN	ProdN	Statement
1	|	|	procedure SumAndProduct(N : integer;
2	|	|	var SumN : integer;
3	|	|	var ProdN : integer);
4	|	|	var
5	|	|	I : integer;
6	|	|	begin
7	|		SumN := 0;
8		|	ProdN := 1;
9	|	|	for I := 1 to N do begin
10	|		SumN := SumN + I;
11		|	ProdN := ProdN * I
12	|	|	end
13	|	|	end;

Figure A.9 Profil de tranche du module SumAnd Product

Les instructions incluses dans la tranche pour C = <13, {SumN}> sont indiquées avec une "|" dans la colonne marquée SumN du profil. La tranche pour C = <13, {ProdN}> est indiquée dans la colonne marquée ProdN.

Les auteurs attirent l'attention du lecteur sur le fait que pour déterminer facilement le degré de cohésion d'un module en se basant sur ses profils de tranches, il faut bien choisir les types d'instructions à inclure dans la tranche, car, par exemple, l'introduction des déclarations ne fait que rendre l'intersection entre les différentes tranches importante, ce qui peut amener à un degré de cohésion erroné. Et, ceci mène à définir « *les instructions exécutables faisant référence à une variable (VRES)* », définies auparavant par Emerson (1984).

Les profils de tranches peuvent indiquer le degré de cohésion d'un module. Par exemple, le module de la figure ci-dessous est de cohésion faible, puisqu'il se compose de trois éléments de traitement distincts sans aucune relation entre eux.

Line	SumN	SumSqrN	ProdN	Statement
				procedure SumAndProduct1(N : integer;
				var SumN : integer;
				var SumSqrN : integer;
				var ProdN : integer);
				var
				I : integer;
				begin
1	|			SumN := 0;
2	|			for I := 1 to N do
3	|			SumN := SumN + I;
4		|		SumSqrN := 0;
5		|		for I := 1 to N do
6		|		SumSqrN := SumSqrN + I*I;
7			|	ProdN := 1;
8			|	for I := 1 to N do
9			|	ProdN := ProdN * I
				end;

Figure A.10 Profil de tranche pour *SumAndProduct1* dont la cohésion est faible

Chaque élément de traitement se situe dans une structure de contrôle à part, en les mettant dans la même structure on obtient une cohésion de type contrôle (voir la figure ci-dessous).

Line	SumN	SumSqrN	ProdN	Statement
				procedure SumAndProduct2(N : integer;
				var SumN : integer;
				var SumSqrN : integer;
				var ProdN : integer);
				var
				I : integer;
				begin
1	|			SumN := 0;
2		|		SumSqrN := 0;
3			|	ProdN := 1;
4	|	|	|	for I := 1 to N do begin
5	|			SumN := SumN + I;
6		|		SumSqrN := SumSqrN + I*I;
7			|	ProdN := ProdN * I
				end
				end;

Figure A.11 Profil de tranche du module *SumAnd Product2* dont la cohésion est de type de contrôle

Le module de la figure ci-dessous calcule trois (3) sorties : X, SumX et SumSqrX. La sortie X est utilisée dans le calcul des deux autres sorties qui ne sont aucunement liées, donc il y a une certaine dépendance de données entre les trois éléments de traitements de ce module, et par conséquent le module a une cohésion de type cohésion de données.

Line	X	SumX	SumSqrX	Statement
				`procedure ProcessArray( N : integer;`
				`var X : IntVector;`
				`var SumX : integer );`
				`var SumSqrX : integer );`
				`var`
				`I, J : integer;`
				`begin`
1	|	|	|	`for I := 1 to N do begin`
2	|	|	|	`X[I] = 0;`
3	|	|	|	`for J := 1 to I do`
4	|	|	|	`X[I] := X[I] + J`
				`end;`
5		|		`SumX := 0;`
6		|		`for I := 1 to N do`
7		|		`SumX := SumX + X[I];`
8			|	`SumSqrX := 0;`
9			|	`for I := 1 to N do`
10			|	`SumSqrX := SumSqrX + X[I]*X[I]`
				`end;`

Figure A.12 Profil de tranche de ProcessArray dont la cohésion est de type cohésion de données

Le module de la figure ci-dessus calcule deux (2) sorties Max et NextMax. Cette dernière dépend fortement de la première, ce qui rend leurs éléments de traitement liés. Effectivement, on remarque que la tranche de la sortie Max est complétement contenue dans la tranche de la sortie NextMax.

La cohésion du module de la figure suivante est forte.

Line	Max	NextMax	Statement
			`procedure FindLargest( A : RealVector;`
			`N : integer;`
			`var Max : integer;`
			`var NextMax : integer );`
			`var`
			`I : integer;`
			`begin`
1	|	|	`if (A[1] >= A[2]) then begin`
2	|	|	`Max := 1;`
3		|	`NextMax := 2`
			`end`
			`else begin`
4	|	|	`Max := 2;`
5		|	`NextMax := 1`
			`end;`
6	|	|	`I := 3;`
7	|	|	`while (I <= N) do begin`
8	|	|	`if (A[I] >= A[Max]) then begin`
9		|	`NextMax := Max;`
10	|	|	`Max := I`
			`end`
			`else`
11		|	`if (A[I] > A[NextMax]) then`
12		|	`NextMax := I;`
13	|	|	`I := I + 1`
			`end`
			`end;`

Figure A.13 Profile de tranche de *FindLargest* dont la cohésion est forte

A.1.4.2 Références

Dans cet article, les auteurs ont utilisé quinze (15) références, dont :

- Deux livres des années 1970, celui de Yourdon et Constantine (1979) et celui de Boehm (1978)
- Douze (12) articles, dont :
  - Deux (2) des années 1970.
  - Dix (10) articles des années 1980, parmi lesquels un a été synthétisé, celui d'Emerson (1984).
- Une thèse de maîtrise qui porte le nom « *Slice based program metrics* » qui date de l'année 1985.

A.2 Couplage et cohésion : Années 1990-1999

Dans cette section, nous résumons sept (7) articles traitant le couple C&C publiés dans les années 1990. Ces articles sont :

1. « Approches basées sur les règles pour le calcul de la cohésion d'un module »
2. « Métriques fondées sur la méthode de tranchage pour l'estimation de la cohésion »
3. « Les difficultés d'utilisation de la cohésion et du couplage comme indicateurs de la qualité »
4. « Suite de métriques pour la conception orientée objet »
5. « Mesure de la cohésion au niveau de la conception »
6. « Détection du couplage logique en se basant sur l'historique des versions du produit »
7. « Mesure de la cohésion et du couplage : Une approche de théorie de l'information »

A.2.1 Lakhotia (1993) « Approches basées sur les règles pour le calcul de la cohésion d'un module »

Constantine et Yourdon ont présenté dans les années 1970 leurs sept (7) niveaux de cohésion dont chacun est relatif à un principe d'association (section III.5). Un principe d'association est défini par Constantine comme « *la relation existante entre les éléments de traitement d'un module* ». Les auteurs, conscients de l'ambiguïté du terme « module », définissent ce dernier comme une procédure PASCAL.

Yourdon et Constantine (1979) ont choisi d'employer le syntagme générique « élément de traitement » à la place d'énoncé ou d'instruction parce que la cohésion peut aussi être calculée pendant la conception, c'est-à-dire avant que le code soit écrit. Donc les éléments de traitement sont tous les énoncés (appels des procédures compris) mêmes « *s'ils n'ont pas encore été réduits à du code* » (Yourdon et Constantine, 1979).

La quantification[54] de la cohésion (par Yourdon et Constantine) est considérée par les auteurs comme étant trop subjective pour être adoptée, puisque les définitions des termes « principe d'association » et « éléments de traitement » sont restées informelles et donc ouvertes à plusieurs interprétations.

Pour que la cohésion devienne un concept objectivement mesurable, il faut donc que « principe d'association » et « éléments de traitement » soient formellement définis.

Les auteurs font correspondre un élément de traitement à une variable de sortie d'un module qui est définie comme étant « *une variable modifiée dans le module et qui est un paramètre passé par référence ou qui est déclarée à l'extérieur du module.* » Ils justifient cette définition par le fait que la fonctionnalité proprement dite du module est déterminée en termes d'entrées/sorties de ce dernier. Quant au principe d'association « *il est formalisé en utilisant les deux relations : dépendance de contrôle et dépendance de donnée entre les variables* ».

Ce qui les emmène à définir ces deux types de dépendance.

- ***Dépendance de données (x ->c (n1,t) y) :*** soient n1 et n2 des énoncés définissant respectivement les variables x et y, $x \neq y$. La variable y a une dépendance de données à l'égard de x s'il y a une chaîne de définition-utilisée [55] de n1 à n2.

- ***Dépendance de contrôle (x -> c (n1)y) :*** une variable y, a une dépendance de contrôle à l'égard d'une variable x, due à une certaine instruction n1, quand n1 contient un prédicat qui emploie x et il existe une instruction n2, qui définit y et l'exécution de n2 peut-être contrôlée par le succès ou l'échec du prédicat dans n1[56]. Quand n1 est une instruction « *if* » si n2 se trouve dans la partie « *then* » il s'agit d'une

---

[54] Yourdon et Constantine, adoptent cette opinion, et pensent aussi que le fait d'assigner une valeur numérique à chaque niveau de cohésion rend le concept de cohésion lui-même « *suspect* », et « *demandent aux programmeurs et concepteurs d'employer ces nombres avec précaution* ».

[55] « *C'est une liste, de chaque utilisation d'une variable, de toutes les définitions qui atteignent cette utilisation* ».

[56] Dans l'article il est écrit n2 mais il s'agit clairement d'une erreur de frappe.

dépendance de type « contrôle vrai » si n2 est dans la partie « *else* » il s'agit d'une dépendance de type « contrôle faux » d'une instruction. Si n1 est une instruction « *while* » la dépendance est toujours « vrai ».

Voici une synthèse de trois autres définitions de l'article :

- ***Graphe de dépendance des variables (Variable Dependence Graph VDG) :*** Un graphe de dépendance des variables d'un module M, noté VM, est le graphe orienté avec les arêtes typées. Les sommets du graphe sont l'ensemble des variables de M et les arêtes relient deux sommets ayant une dépendance de type contrôle ou de type données entre eux[57].

- ***Variable à un seul but :*** on dit qu'une variable est à un seul but, si toutes ses définitions atteignent l'extrémité du module.

- ***Module canoniquement nommé :*** est un module dont chaque variable a un seul but.

Le tableau ci-dessous expose les règles proposées par les auteurs pour déterminer la cohésion d'une paire de variables de sortie et leurs explications telles que présentées dans l'article. Ces règles sont sous forme d'expressions logiques : $rule_i$ (i=1...5), la cohésion de deux variables x et y est Ci si et seulement si $rule_i$ (x,y) = vrai.

---

[57] Voici la définition formelle : (VM) ={e tel que e= x -> d y avec :
x, y ∈ v(VM) et y a une dépendance de données avec x en M, ou e= x -> c(n,k) y avec :
x, y ∈ v(VM) et y a une dépendance de contrôle de type k avec x du à n en M.}
v(VM) = Var(M) (l'ensemble de variables de module M).

Tableau A.5 Les cinq règles de calcul des cinq degrés de cohésion

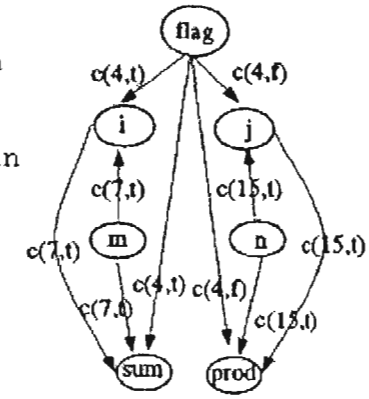
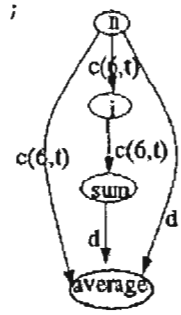
Cohésion	Règle	Explication
1. Accidentelle	$rule_1(x, y) = \neg(\bigvee_{\forall i, i \in \{2 \ldots 5\}} rule_i xy)$	Deux variables qui n'ont ni une cohésion logique, ni procédurale, ni communicationnelle ni séquentielle ont une cohésion accidentelle[58].
2. Logique	$rule_2(x, y) = \exists znk.\forall l.$ $z \rightarrow_{c(n,k)} x \wedge z \rightarrow_{c(n,\neg k)} y$ $\wedge\neg(z \rightarrow_{c(n,l)} x \wedge z \rightarrow_{c(n,l)} y)$	Deux variables ont une cohésion logique si elles ont deux types différents de dépendance de contrôle à l'égard d'une même variable au même nœud.
3. Procédurale	$rule_3(x, y) = (\exists znk.z \rightarrow_{c(n,k)} x \wedge z \rightarrow_{c(n,k)} y)$	Deux variables ont une cohésion procédurale si elles ont une dépendance de contrôle du même type à l'égard d'une même variable au même sommet. Ceci est vérifiable si les définitions des deux variables apparaissent dans une même instruction « tant que » ou une même instruction « si ».
4. Communicationnelle	$rule_4(x, y) = \exists z.\forall nkl.$ $\neg(z \rightarrow_{c(n,k)} x \wedge z \rightarrow_{c(n,\neg k)} y)$ $\wedge\neg(z \rightarrow_{c(n,k)} x \wedge z \rightarrow_{c(n,k)} y)$ $\wedge(\ (z \rightarrow x \wedge z \rightarrow y)$ $\vee(x \rightarrow z \wedge y \rightarrow z))$	Deux variables ont une cohésion communicationnelle si elles ont une relation entre elles ou une relation avec une même variable, et cette relation ne correspond pas aux cohésions logiques et procédurales. Si on veut détailler : Elles ont une dépendance de données à l'égard d'une même variable, ou une même variable a une

[58] Il est intéressant de noter que Yourdon et Constantine (1979) définissent la cohésion fonctionnelle par la négative et non la cohésion accidentelle comme l'auteur de cet article.

Cohésion	Règle	Explication
		dépendance de données à l'égard d'eux, ou l'une d'elles a une dépendance de contrôle et l'autre a une dépendance de données à l'égard d'une même variable Les deux variables ont une dépendance de contrôle à l'égard d'une même variable
5. Séquentielle	$rule_5(x, y) = (x \rightarrow_d y \vee y \rightarrow_d x)$	Deux variables ont une cohésion séquentielle si l'une a une relation de dépendance de données vis-à-vis de l'autre.

Dans le tableau ci-dessous, on considère des exemples de modules avec leurs graphes de dépendance de variables.

Tableau A.6 Exemples de modules avec leurs graphes de dépendance de variables

Module avec GDV :	Analyse de la cohésion du module :
1 procedure sum_or_product(m,n,flag: integer; var sum,prod: integer); 2 var i,j: integer; 3 begin 4 if flag = 1 then begin 5 i := 1; 6 sum := 0; 7 while i <= m do begin 8 sum := sum + i; 9 i := i + 1 10 end 11 end 12 else begin 13 j := 1; 14 prod := 1; 15 while j <= n do begin 16 prod := prod * j; 17 j := j + 1 18 end 19 end 20 end 	Ce module calcule la somme (sum) ou le produit (prod) des n premiers entiers. Le GDV, montre que les deux variables de sortie sum et prod, ont une dépendance de contrôle à l'égard de la variable flag (instruction de la ligne 4), et cette dépendance est de différents types : vrai pour sum et faux pour prod. Donc, ce module a une cohésion logique, puisqu à chaque invocation, c'est seulement l'une des deux fonctions qui est exécutée.
1 procedure sum_and_average(n:integer; var sum,average: integer); 2 var i: integer; 3 begin 4 i := 1; 5 sum := 0; 6 while i <= n do begin 7 sum := sum + i; 8 i := i + 1 9 end; 10 average := sum / n 11 end 	Ce module calcule la somme (sum) et la moyenne (average) des n premiers entiers. Le calcul de la variable de sortie average utilise la variable de sortie sum, donc average a une dépendance de donnée à l'égard de sum. Ce module est de cohésion séquentielle.

L'auteur se base sur l'algorithme SMC (*Compute-Module-Cohesion*) de la méthode de calcul de cohésion de Stevens, Myers et Constantine (1974) pour proposer un algorithme

formalisant son procédé de calcul avec quelques considérations personnelles. En gros il procède comme suit :

La cohésion du module étudié est :

- *fonctionnelle* si ce module comprend une seule variable de sortie,
- *indéfinie* si le module ne contient aucune variable, autrement
- la plus petite cohésion des paires de variables de sortie du module.

Cet algorithme et l'algorithme de Stevens, Myers et Constatine sont présentés dans la figure ci-dessous.

Algorithme de l'auteur	Algorithme de Stevens et al.
Algorithm *AL-compute-module-cohesion* ***Input:*** *a canonically named module P* ***Output:*** *Module cohesion of P or 'undefined'* *Construct the PDG of P* *Construct the VDG V_P* *Let X be the set of all the output variables of P* **If** $\mid X \mid = 0$ **then** *cohesion := 'undefined'* **else if** $\mid X \mid = 1$ **then** *cohesion := 'functional'* **else begin** *- - initialize to highest value* *cohesion := 'sequential'* **for** x **in** X **and** y **in** Y **and** $x \neq y$ **do** *- - maximum of all cohesions for a variable pair* $VC := MAX(\{C_i \mid i \in \{1..5\} \wedge rule_i(x, y)\})$ *- - minimum of all pairs* *cohesion := min(cohesion, VC)* **end-for** **end** **return** *cohesion* ***end***	**Algorithm:** *SMC's-compute-module-cohesion* **Input:** *A module or its narrative description* **Output:** *Cohesion of the module* *Identify the set of processing elements of the module* *For every pair of processing elements do* • *identify the set of associative principles in Table 1 that suitably define the association between the pair* • *the highest level of cohesion corresponding to these principles is the cohesion for the pair* *The cohesion of a module is the lowest cohesion of all pairs of processing elements of the module.*

Figure A.14 Algorithme de calcul de cohésion

Enfin, l'auteur effectue une comparaison de sa méthode à celle d'Ott et Thuss (1989) et celle d'Emerson (1984). Avant d'entamer cette étude comparative, l'auteur décrit ces deux approches.

Approche d'Ott et Thuss : Dans cette approche, le type de relation entre les éléments de traitement du programme est défini en examinant les instructions dans l'intersection des tranche-fin[59] de ses variables de sortie[60]. Ils (Ott et Thuss) classent les sept (7) niveaux de cohésion en quatre (4) catégories présentées dans le tableau ci-dessous.

Tableau A.7 Catégories de cohésion d'Ott et Thuss (1989)

Cohésion	Principe d'association d'Ott et Thuss
Faible (Accidentelle, temporelle)	L'intersection est vide.
Contrôle (Logique, Procédurale)	l'intersection contient principalement des instructions de contrôle et des définitions de variables de contrôle.
Donnée (Communicationnelle)	L'intersection contient des définitions de données de variable non-contrôle.
Haute (Séquentielle, Fonctionnelle)	Une tranche est complètement contenue dans l'autre.

Approche d'Emerson : Il représente un programme comme un graphe de flux et construit l'ensemble de références[61] de chaque variable.

---

[59] « *Une tranche-fin d'une variable de sorite est une tranche exécutée à la dernière instruction en ce qui concerne cette variable* ».

[60] Selon leur définition, une variable de sortie est un paramètre de référence ou une variable définie par le système d'exploitation.

[61] L'ensemble de sommets qui se rapportent à cette variable dans le graphe de flux.

Si Ri est l'ensemble de sommets qui font référence à la variable i dans le graphe de flux F, il définit la métrique k (Ri, F) comme suit :

$$k(R_i, F) = \frac{|R_i| dim R_i}{|\nu(F)| dim\ \nu(F)}$$

Avec, dimA : le nombre « maximal de chemins linéairement indépendants » passant par l'ensemble de sommets de A dans le graphe de flux F.

Ensuite, il considère la cohésion du module F par k(F), comme la moyenne arithmétique de k(Ri, F) pour toutes les variables i. On appelle cette mesure, la cohésion du graphe.

Il classifie les 7 niveaux de cohésion ordinaires en trois (3) types de cohésion, définis comme suit :

- ***Type 1 :*** $0 \leq k(F) \leq (q/c)/s$ => {fonctionnelle, séquentielle, communicationnelle}
- ***Type 2 :*** $(q/c)/s \leq k(F) \leq 1/s$ => {procédurale, temporelle}
- ***Type 3 :*** $1/s \leq k(F) \leq 1$ => {logique, fortuite[62]}

Avec :

- S[63] : le nombre d'instructions qui se rapportent à des variables.
- Q : le nombre moyen de variables référencées par instruction exécutable.
- C : le rapport du nombre de variables dans le module et du nombre d'instructions exécutables.

---

[62] Fortuite est synonyme d'accidentelle.

[63] « *Les constantes q et c sont des paramètres spécifiques du langage de programmation censés être calculé à partir d'un échantillon des programmes* ».

Par la suite, l'auteur choisit sept (7) modules, calcule leur cohésion selon son approche, selon l'approche de Stevens, Myers et Constantine, selon celle d'Emerson, et enfin selon celle d'Ott et Thuss. C'est ce qui lui donne le tableau ci-dessous :

(*) : Indique que la cohésion assignée à ce module diffère de celle assignée par Stevens, Myers et Constantine.

(?) : indique que la classification de ce module est ambigüe ou mal définie.

Tableau A.8 Résultats de la comparaison

Approach → Module ↓	Stevens et. al.'s	Lakhotia & Nandigam's	Emerson's	Ott & Thuss'
Module 1	functional	functional	Type I	High
Module 2	logical	logical	Type III	Control
Module 3	communicational	communicational	Type II (*)	Low (*)
Module 4	procedural	procedural	Type I (*)	Control
Module 5	sequential	sequential	Type II (*)	High
Module 6	coincidental	coincidental	Type II (*)	Low
Module 7	logical	logical	Type III	?

On remarque qu'il n'y a aucune concordance dans les résultats de la méthode de calcul de cohésion de l'auteur et celle d'Emerson. Ce dernier modélise les modules de type 2, « *comme des modules dont le graphe de flux peut être construit par une séquence de graphes de flux plus simples* ». Et les modules de type 3, « *comme des modules dont le graphe de flux, est construit d'une séquence de graphes de flux parallèlement connectés de telle sorte qu'un seul d'entre eux sera sélectionné pour être exécuté* ». Selon l'auteur, cette modélisation est incorrecte, du moment qu'il y a des modules dont la construction correspond parfaitement à celle du type 2 ou 3, mais dont le degré de cohésion n'est pas de type 2 ou 3. Donc, Lakhotia trouve que la base de cette classification de la cohésion induit en erreur. Quant à la méthode d'Ott et Thuss, elle catégorise un module de cohésion communicationnelle comme un module

de cohésion faible parce que l'intersection des profils de tranches des variables de sortie est vide, ce qui est inacceptable.

A.2.1.1 Références

L'auteur a utilisé quatorze[64] (14) références :

- Cinq livres, un des années 1980, et trois des années 1970, parmi lesquels il y a le livre de conception structurée de Yourdon et Constantine (édition 1978) et le livre de Stevens, Myers et Constantine (1974).
- Huit articles, deux des années 1970, trois des années 1980, deux d'entre eux ont été synthétisés et analysés : Emerson (1984) et Ott et Thuss (1989)
- Un rapport technique.

A.2.2 Ott et Thuss (1993) « Métriques fondées sur la méthode de tranchage pour l'estimation de la cohésion »

En 1981, Weiser s'était basé sur l'organisation en tranches d'un programme pour définir de nouvelles métriques de complexité aussi performantes que celle de McCabe et Halstead (Weiser, 1981). Ces métriques étaient au nombre de cinq (5) :

1. Couverture (coverage)
2. Chevauchement (overlap)
3. Rigidité (tightness)
4. Parallélisme (parallelism)
5. Groupement (clustering)

---

[64] Il n'Existe que quatorze (14) références même si l'auteur écrit qu'il y en a quinze (15).

Les auteurs citent la thèse de H. D. Longworth (1985) où celui-ci a montré l'existence d'une relation entre la cohésion et trois de ces métriques (couverture, chevauchement et rigidité).

Dans l'article que nous analysons les auteurs étudient :

- Les trois métriques analysées par Longworth.
- Une métrique provenant de l'article de Weiser (parallélisme).
- Deux nouvelles métriques, MinCoverage et MaxCoverage.

Voici les définitions de ces six (6) métriques.

Coverage : est défini comme étant, la longueur moyenne des tranches divisée par la longueur du module

$$Coverage(M) = \frac{1}{|Vo|} \sum_{i=1}^{|Vo|} \frac{|SL_i|}{\mathbf{length}(M)}$$

Avec :

- M : le module.
- Vo : l'ensemble de variables de sortie du module M.
- Sli: la tranche de la variable vi ∈ VRES (*Variable-Referent Executable Statements*). VRES étant le nombre d'instructions exécutables et d'expressions qui font référence à au moins une variable d'un module.
- Length(M) : la longueur du module M qui est égale à VRES.

Overlap : c'est le rapport moyen du nombre d'instructions de l'intersection de toutes les tranches et de la taille de chaque tranche.

$$Overlap(M) = \frac{1}{|Vo|} \sum_{i=1}^{|Vo|} \frac{|SL_{int}|}{|SL_i|}$$

Avec, Slint, l'intersection de toutes les tranches Sli.

$$\mathbf{SL}_{int} = \bigcap_{i=1}^{|V_O|} \mathbf{SL}_i.$$

Tightness : c'est le rapport du nombre d'instructions de l'intersection de toutes les tranches et de la longueur du module.

$$Tightness(M) = \frac{|SL_{int}|}{\text{length}(M)}$$

Parallelism : le parallélisme indique le nombre de tranches qui ont peu de choses en commun avec toutes les autres tranches. Il est exprimé comme le nombre de tranches ayant par paires un chevauchement avec toutes les autres tranches inférieur ou égal à un seuil ε.

Parallelism(M)= | { Sli tel que : |Sli ∩ Slj| ≤ ε pour tout : j ≠ i }|

Si ε = 0, alors parallelism détermine le nombre de tranches qui sont complètement indépendantes de toutes les autres tranches du module.

MinCoverage : le rapport de la longueur de la plus petite tranche dans le module et la longueur du module.

$$MinCoverage(M) = \frac{1}{\text{length}(M)} \min_i |SL_i|$$

MaxCoverage : le rapport de la longueur de la plus grande tranche dans le module et la longueur du module.

$$MaxCoverage(M) = \frac{1}{\text{length}(M)} \max_i |SL_i|$$

En appliquant ces métriques, Longworth s'est rendu compte qu'elles étaient sensibles à la taille et l'emplacement des éléments de traitements d'un module. Pour comprendre le

problème, considérons les programmes des deux figures suivantes (Figure A.15 et Figure A.17).

Les programmes de la Figure A.15 sont utilisés pour montrer l'influence des emplacements des éléments de traitement sur les métriques étudiées. Les deux programmes comprennent trois (3) éléments de traitement qui calculent les variables : **a**, **b**, et **c**.

Le premier programme (*Input-Sided*) calcule **a** et **b** qui ne dépendent pas l'une de l'autre mais dépendent d'une même entrée **x**, et sont utilisées dans le calcul de **c**. Le deuxième programme (*Output-sided*) calcule **a** qui est requise pour les éléments de traitement de calcul de **b** et **c**. Ces deux programmes sont de cohésion communicationnelle, puisque leurs éléments de traitements sont liés entre eux par des données. Donc, ils ont le même type de relation et (presque) le même degré de dépendance entre leurs éléments de traitement. Mais, les auteurs ont remarqué une assez importante différence au niveau des valeurs des métriques *Overlap*, *Tightness* et *MaxCoverage* de ces deux programmes.

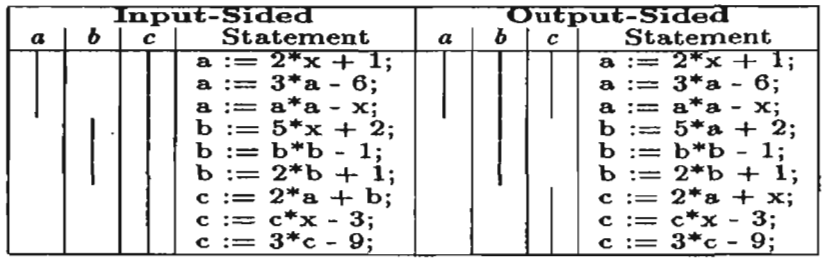


Figure A.15 Profils de tranches des programmes *Input-sided* et *Output-sided*

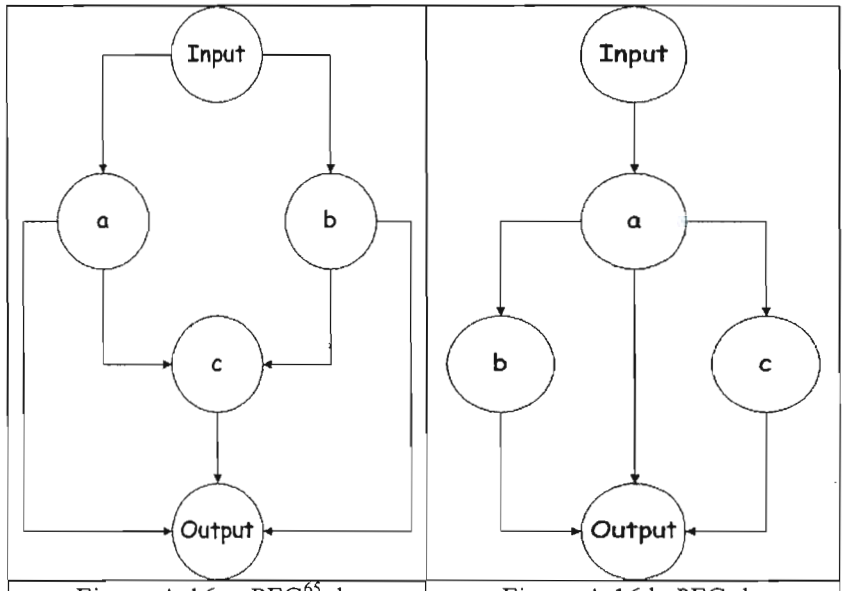


Figure A.16.a PFG[65] du programme *Input-sided*

Figure A.16.b PFG du programme *Output-sided*

Figure A.16 PFG des programmes *Input-Sided* et *Output-Sided*

Dans le premier programme (*Small-Large*) les deux premiers éléments de traitement qui calculent les variables **a** et **b**, sont très petits, alors que le troisième qui calcule la variable **c**, est considérablement plus grand par rapport à eux. Alors que dans le deuxième programme (*Large-Small*), le premier élément de traitement qui calcule la variable **a** est grand, et les deux derniers qui calculent les variables **b** et **c** sont très petits. Ces deux programmes ont le même degré de cohésion (cohésion séquentielle), mais le tableau ci-dessous, montre qu'il y a une différence entre les valeurs de métriques *Coverage*, *Overlap*, *Tightness* et *MinCoverage* des deux programmes.

[65] Graphe de Flux d'éléments de Traitement (*Processing element Flow Graph*).

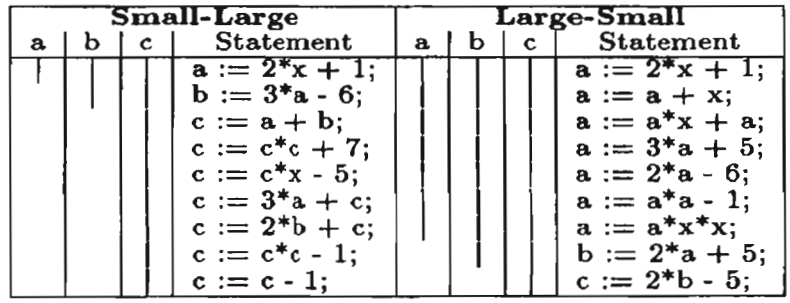


Figure A.17 Profils de tranches des programmes *Small-Large* et *Large-Small*

Tableau A.9 Les métriques de tranche basées les profils communicationnel et séquentiel des figures précédentes (pour montrer l'influence de l'emplacement et la taille des éléments de traitement)

Module	*Coverage*	*Overlap*	*Tightness*	*MinCoverage*	*MaxCoverage*	*Parallelism*
Input-Sided	0.56	0.00	0.00	0.33	1.00	0
Output-Sided	0.56	0.67	0.33	0.33	0.67	0
Small-Large	0.44	0.53	0.11	0.11	1.00	0
Large-Small	0.89	0.88	0.78	0.78	1.00	0

Pour remédier à ce problème, les auteurs ont remarqué qu'en examinant les tranches pour estimer le degré de cohésion, on s'intéresse aux instructions qui affectent une variable donnée et aussi aux instructions influencées par cette variable. Donc, ils ont défini des « *metric slices* » qui prennent en considération les deux types de relations et cela en combinant les tranches d'avant et d'arrière. Ces tranches permettent en quelque sorte de réduire la sensibilité des métriques étudiées vis-à-vis de la taille et de l'emplacement des éléments de traitement.

Les auteurs définissent « *metric slice* » comme, « *l'union des instructions de la tranche en arrière (backward slice) et les instructions des tranches en avant pour une variable donnée* ». La tranche en arrière est obtenue à partir du graphe de dépendance en scrutant ce

dernier et dépistant tous les sommets qui ont un effet sur la variable en partant du sommet de cette dernière, jusqu'à l'atteinte du sommet du graphe.

La tranche en avant est calculée toujours en utilisant le graphe de dépendance, en décelant tous les sommets qui sont influencés par la variable de la tranche prenant comme départ le sommet de cette dernière, jusqu'à atteinte des sommets du dernier niveau du graphe.

On présente maintenant les profils de tranche des deux programmes avec application des *metric slices* dans la Figure A.18. On remarque que les profils des deux programmes *Input-Sided* et *Output-Sided* sont devenus symétriques. Aussi, les valeurs des métriques de ces deux programmes présentées dans le tableau A.10 sont identiques.

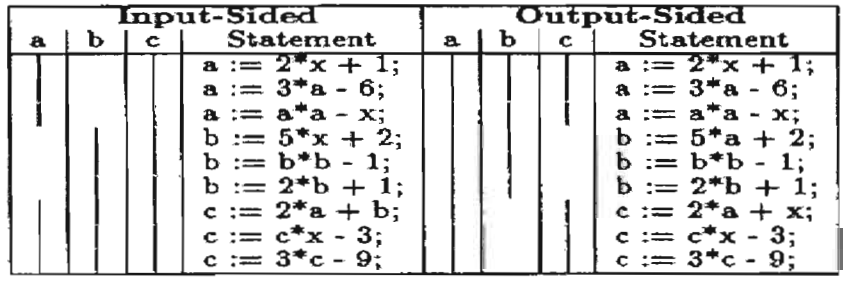


Figure A.18 Profils de *metric slices* des programmes *Input-Sided* et *Output-Sided*

Tableau A.10 Les métriques basées sur les tranches des modules étudiés

Module	*Coverage*	*Overlap*	*Tightness*	*MinCoverage*	*MaxCoverage*	*Parallelism*
Input-Sided	0.78	0.44	0.33	0.67	1.00	2
Output-Sided	0.78	0.44	0.33	0.67	1.00	2
Small-Large	1.00	1.00	1.00	1.00	1.00	0
Large-Small	1.00	1.00	1.00	1.00	1.00	0

Enfin, nous arrivons à adopter ces métriques comme des indicateurs du degré de cohésion, le tableau ci-dessous montre la correspondance du degré de cohésion et les valeurs des métriques.

Tableau A.11 Degrés de cohésion et valeurs des métriques

Cohésion	Métriques					
	Coverage	Overlap	Tightness	Min-Coverage	Max-Coverage	Parallelism
Forte	1	1	1	1	1	0
Faible	0	0	0	0	0	!=0
Acceptable	]0,1[	]0,1[	]0,1[	]0,1[	]0,1[	]0,1[

A.2.2.1 Références

Les auteurs ont utilisé vingt-sept (27) références :

- Cinq livres, deux des années 1970, dont un est celui de Yourdon et Constantine (1979), deux des années 1980 et un des années 1990.
- Dix-huit articles, douze articles des années 1980, dont deux ont été synthétisés : Emerson (1984) et Ott et Thuss (1989) et un consulté (weiser (1981)).
- Trois rapports techniques.
- Un mémoire de maîtrise de Longworth (1985).

A.2.3 Woodward (1993) « Les difficultés d'utilisation de la cohésion et du couplage comme indicateurs de la qualité »

L'auteur commence par donner les définitions et les échelles traditionnelles[66] de la cohésion et du couplage, en attirant l'attention du lecteur, qu'en littérature, on trouve quelquefois certaines « variations mineures[67] » dans ces échelles.

---

[66] Celles présentées par Yourdon et Constantine (1979).

[67] Comme Myers qui omet la cohésion séquentielle et le couplage STAMP, et Jones qui ajoute une catégorie de cohésion nommée cohésion informationnelle entre la fonctionnelle et la séquentielle, et Fenton qui omet le couplage externe.

Ces catégorisations, avec leurs descriptions, ont été mises en œuvre « *en se basant sur les langages de programmation des années 1970 comme Fortran [...] il n'est pas du tout clair comment ces concepts sont en relation avec, par exemple, les concepts de paquets, les types génériques et privés dans les langages procéduraux [...] elles semblent ne pas dire grand chose pour les programmes de nature déclarative [...] Le besoin d'une évaluation des idées pour faire face à la diversité des langages est ressenti...et cela n'est pas du tout nouveau car Yourdon et Constantine ont bien exprimé cela en 1979, en définissant le processus de catégorisation comme "un outil inachevé en cours de développement évolutionnaire*" ».

Le programme étudié est un utilitaire de restructuration des programmes Fortran en ajoutant des indentations selon les indications d'un manuel Fortran. Il est constitué d'un programme principal (*INDENT*), et cinq autres sous-programmes. Ci-dessous, on présente les fonctions de chacun de ces composants :

INDENT : « *Un programme pour restructurer des programmes de Fortran en dentelant le corps de toutes les BOUCLES-DO et les blocks-IF par des colonnes d'ISHIFT. Ceci permet une inspection visuelle rapide pour les parties les plus profondément indentées d'un programme.* »

- *PUNITS* : Le sous-programme principal de *INDENT* qui vérifier les entrées.
- *REFORM* : Restructure des instructions acceptées.
- *IDENT* : Traitement de *BOUCLE-DO*, *BLOC-IF* et FORMAT.
- *START* : Préparation pour *PUNITS*.
- *TERMIN* : Afficher le résultat final.

Ces programmes ont été présentés à 163 étudiants[68] afin de les étudier et de les catégoriser selon leur niveau de cohésion et de couplage. L'ensemble du programme compte 621 lignes de code, dont 217 sont des lignes de commentaires.

La figure ci-dessous montre comment ces différents modules interagissent entre eux.

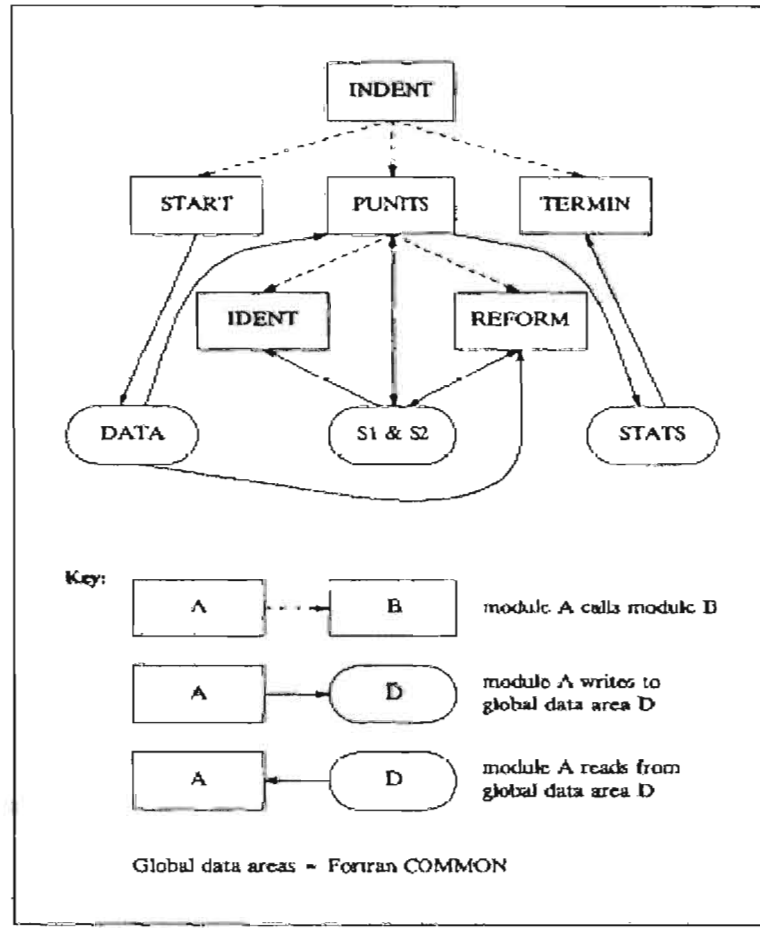


Figure A.19 Conception architecturale du programme Fortran *INDENT*

[68] On note que ces étudiants n'ont aucune expérience avec Fortran, mais ils ont été initiés à ce langage afin d'accomplir la tâche qui leur a été assignée.

Maintenant, on présente les résultats de catégorisation de cohésion, donnés par les 163 étudiants. À première vue, on remarque (voir le tableau ci-dessous) que chaque catégorie de cohésion a été choisie par au moins un étudiant pour chaque module, à l'exception des modules *REFORM* et *PUNITS*.

Tableau A.12 Catégorisation de la cohésion des modules par les 163 étudiants

	INDENT	***PUNITS***	***REFORM***	***IDENT***	***START***	***TERMIN***
Fonctionnelle	4	0	16	26	28	89
Séquentielle	26	45	62	7	27	2
Communicationnelle	1	45	50	58	19	5
Procédurale	41	34	17	17	4	2
Temporelle	3	5	0	2	72	48
Logique	4	23	8	46	6	5
Accidentelle	1	8	4	2	2	4
Pas de catégorie	83	3	6	5	5	8

Le tableau ci-dessous présente une analyse détaillée de la cohésion des différents modules.

Tableau A.13 Étude de la cohésion des modules avec le pourcentage des étudiants qui sont arrivés à l'identifier correctement

Module	**Analyse du module:**	**Analyse de cohésion du module:**	**Pourcentage d'étudiants trouvant la bonne cohésion:**
INDENT	Il contient trois instructions exécutables, qui sont des appels des modules START, PUNITS et TERMIN, en respectant l'ordre des appels.	Procédural, puisqu'il y a un ordre d'exécution des éléments du module sans aucune connexion de donnée entre eux.	25.15%
PUNITS	Ce module lit les lignes du fichier d'entrée. Il accumule les lignes de commentaires dans un buffer, toutes les instructions de	Les activités diversifiées de ce module qui agissent sur la même donnée peuvent nous laisser penser qu'il a une cohésion	20.86%

Module	Analyse du module:	Analyse de cohésion du module:	Pourcentage d'étudiants trouvant la bonne cohésion:
	Fortran, qui peuvent être éparpillées dans plusieurs lignes d'entrée dans un autre buffer.	communicationnelle, mais l'ordre de ces activités essentiel, le rend d'une cohésion procédurale.	
REFORM	Ce module exécute deux activités majeures qui sont la restructuration des instructions et l'affichage de ces instructions structurées. Il comprend aussi une activité mineure qui est l'affichage des commentaires qui suivent les instructions structurées.	Si on ne prête pas attention à l'activité mineure du module, on peut affirmer que ce module est de cohésion séquentielle, mais la présence de cette activité rend sa cohésion communicationnelle, puisque toutes ses activités agissent sur la même donnée.	30.67%
START	Les activités de ce module, correspondent à l'initialisation des valeurs des variables, leur vérification, après leur affichage.	Le fait que ces mêmes activités sont une forme d'une séquence de transformations de données pouvant être des entrées de la prochaine activité peut nous amener à dire qu'il est de cohésion séquentielle. Mais, ces activités doivent être complètes avant la lecture et le traitement du programme Fortran, donc il est d'une cohésion temporelle.	44.17%
TERMIN	Il contient une seule instruction exécutable, l'affichage de l'information statistique du programme Fortran qui a été dentelé.	Puisque ce module a une seule fonction, donc on peut se tromper et dire qu'il a une cohésion fonctionnelle, mais on oublie que cette fonction peut dans certains cas ne jamais être effectuée. Mais, ce qui est certain, c'est qu'elle est la dernière tâche du module, donc ce dernier, est de cohésion temporelle.	29.45%

Le tableau ci-dessous présente une analyse détaillée du couplage des différents modules.

Tableau A.14 Étude du couplage des modules avec pourcentage des étudiants qui sont arrivés à l'identifier correctement

Paire de modules	Analyse du couplage	Pourcentage d'étudiants qui ont choisi le bon couplage
INDENT & START PUNITS TERMIN	Bien qu'il y ait un couplage entre INDENT et ces trois modules via les appels de ces derniers par INDENT, comme on l'a déjà mentionné, il est faible et il ne correspond à aucune catégorie de couplage parmi les catégories standardisées connues. Donc il n'y a pas un couplage direct entre eux.	17.17%
PUNITS & IDENT	Ces deux modules partagent l'accès à des données communes S1 et S2, donc ils ont un couplage commun, et PUNITS appelle IDENT, avec des fanions comme des paramètres courants, indiquant le point de retour (return) dans le module appelant IDENT. Ces paramètres n'affectent en aucun cas l'exécution du module appelé. Ils ont un couplage de données ou de contenu	1ère catégorie: 76% 2ème catégorie: 27%
PUNITS& REFORM	Ces deux modules exhibent eux aussi deux types de couplage : le couplage commun, puisqu'ils partagent l'accès à trois blocs communs, S1, S2 et DATA ; le deuxième couplage est le couplage de contrôle, car PUNITS appelle REFORM, avec un seul paramètre logique FORM, qui indique si l'instruction est de format Fortran ou non.	1ère catégorie: 73.62% 2ème catégorie: 51.53%
START & PUNITS	Ils partagent l'accès au même bloc DATA, donc ils ont un couplage commun.	40%
START & REFORM	Ils partagent l'accès au même bloc DATA, donc ils ont un couplage commun.	23%
PUNITS & TERMIN	Ils partagent l'accès au même bloc STATS, donc ils ont un couplage commun.	37%
IDENT & REFORM	Ces deux modules partagent l'accès à des blocs communs nommés S1 et S2, donc ils ont un couplage commun.	23%

Les résultats de cette étude de cas, mènent l'auteur à déduire que la catégorisation de la cohésion et du couplage d'un module donné, est « *un processus difficile dont la bonne application nécessite une expérience et une performance importantes, donc il est fort souhaitable d'automatiser ce processus* », pour qu'il soit plus fiable.

L'auteur souligne l'importance de l'approche de tranchage (Weiser 1981, Ott et Thuss 1989, voir A.1.4) et de celle d'Emerson (Emerson, 1984, voir A.1.1).

Comme exemples de métriques de couplage l'auteur cite, la métrique de Fenton et Melton (1991), appelée C, et qui mesure le couplage entre les modules M1 et M2, en prenant en considération le fait qu'il peut y avoir plusieurs types de couplage entre ces modules. La métrique présentée est la suivante :

C (M1, M2)= i + n/(n+1)

Avec:

- I : le poids du plus faible couplage entre M1 et M2.
- N : le nombre d'interconnexions entre M1 et M2. L'auteur signale l'ambigüité de ce « n », qu'on ne sait pas s'il correspond au nombre d'interconnexions du couplage I, ou de tous les couplages de M1 et M2.

Mais, cette métrique reste tout de même subjective, puisque c'est à nous d'identifier le ou les type(s) de couplage entre M1 et M2.

Et, bien avant, Henry et Kafura (1981) avaient proposé une métrique de couplage qui se basait sur le flux inter-modules de l'information, qui peut identifier deux catégories de couplage : le « couplage contenu » qui équivaut dans ce cas à l'existence de flux locaux directs[69], et le « couplage commun », qui lui aussi, selon cette approche, correspond à l'existence de flux globaux[70].

La façon avec laquelle ils mesurent la force de connexion de deux modules A et B est la suivante:

---

[69] C'est-à-dire l'appel d'un module par un autre.
[70] C'est-à-dire l'utilisation par un module de l'information déposée par autre module dans une structure de données globale.

(Nombre de modules exportant de l'information de A + Nombre de modules important de l'information à A) * Nombre de chemin d'information[71]

Et en 1992, Shepperd (1992), est venu améliorer cette métrique, qui est devenue :

$$C = \sum_{i=1}^{n} C_i$$

Avec:

- N : le nombre de modules.
- Ci : le couplage du $i^{ème}$ module.
- Ci = FAN_INi *FAN_OUTi, où :
  - FAN_Ini: nombre de modules qui écrivent dans n'importe quelle structure de données que le $i^{ème}$ module lit à partir d'elle, sauf les écritures du $i^{ème}$ module lui même.
  - FAN_OUTi: nombre de modules qui lisent de n'importe quelle structure de données dans laquelle le $i^{ème}$ module écrit sauf les lectures du $i^{ème}$ module.

A.2.3.1 Références

L'auteur a utilisé vingt-trois (23) références :

- Dix livres, le livre Yourdon et Constantine (1979) et le livre de Stevens, Myers et Constantine (1974), quatre qui datent des années 1980 et quatre des années 1990, parmi eux celui de Pressman (1992).
- Treize articles :
  - Un des années 1970, il s'agit du fameux article de Myers (1973).

---

[71] « *Le nombre de chemins d'information doit tenir compte des flux directs et indirects d'A à B via les modules intermédiaires* ».

- Huit des années 1980, parmi eux nous trouvons les articles d'Emerson (1984) et d'Ott et Thuss (1989).
- Quatre des années 1990.

A.2.4 Chidamber et Kemerer (1994) « Suite de métriques pour la conception orientée objet »

Le problème de recherche soulevé par les auteurs est que, à cause de l'énorme succès du développement OO, il est important de proposer des métriques qui tiennent compte des caractéristiques de cette approche. D'un point de vue plus général, les auteurs considèrent aussi que les métriques disponibles à cette époque n'avaient pas une base théorique solide. Effectivement plusieurs auteurs se sont intéressés à cela comme :

- Morris qui a présenté quelques métriques qui n'ont pas été étudiées expérimentalement (Morris, 1988).
- Moreau et Dominick (1989) qui ont proposé trois métriques pour les systèmes d'information graphique OO, mais ils n'ont pas fourni des définitions formelles et examinables.
- Chidamber et Kemerer (1991), Sheetz (1992), et Whitmire (1992) qui ont proposé des métriques, mais sans aucune donnée empirique.

Dans toutes ces tentatives on n'a jamais présenté des données empiriques permettant de vérifier les métriques en les appliquant sur des projets OO. Il est donc fondamental selon les auteurs de proposer une suite théoriquement solide et empiriquement vérifiable.

A.2.4.1 Base théorique des métriques de conception orientée objet (COO)

Parmi les nombreuses méthodes de conception OO, les auteurs ont choisi celle de Booch (1991), dont la première priorité est la conception de classe. Donc, les métriques proposées concernent la complexité des classes plutôt que celle des systèmes. Cette approche a comme point faible, « *la non capture du comportement dynamique du système* ».

Le processus de cette méthodologie est constitué de quatre (4) activités principales :

5. *Identification des classes (et des objets)*

6. *Identification de la sémantique des classes (et des objets)*
7. *Identification des relations entre les classes (et les objets)*
8. *Mise en œuvre des classes.*

Les principes ontologiques proposés par Bunge (1997) constituent le fondement du concept objet, et ont été la base des définitions des concepts du domaine OO.

En effet, un objet peut être représenté comme suit :

X=(x, p(x)) où : x est l'individu substantiel et p(x) est la collection finie de ses propriétés.

On juge en termes ontologiques que deux objets sont couplés si « *au moins l'un d'entre eux agit sur l'autre, on dit que X agit sur Y si l'historique de Y est affectée par X, où l'historique est définie comme les états classés chronologiquement qu'une chose traverse à travers le temps* ».

Soient deux objets : X=(x, p(x)) et Y=(y, p(y))

P(x)= {Mx} □ {Ix}

P(y)={My} □ {Iy}

Où: {Mi} est l'ensemble des méthodes et {Ii} l'ensemble de variables d'instance de l'objet i.

« *Bunge définit la cohésion c de deux choses comme l'intersection des ensembles de propriétés de ces deux choses* ».

$C(X,Y)= p(x) \cap p(y)$

Bunge définit la complexité d'un individu comme le nombre d'éléments de sa composition, impliquant qu'un individu complexe a un grand nombre de propriétés.

En utilisant cette définition comme base, la complexité d'un objet d'une classe peut être définie comme la cardinalité de son ensemble de propriétés. Complexité de (x, p(x)) = l p(x)), où lp(x) est la cardinalité du p(x).

Profondeur d'héritage = la profondeur de la classe dans l'arbre d'héritage.

Nombre d'enfants = le nombre de descendants immédiats de la classe.

Des méthodes peuvent être vues comme des définitions de réponses possibles aux messages. Donc, on définit une réponse réglée pour une classe d'objets de la façon suivante :

Ensemble de réponses des objets d'une classe = {ensemble de toutes les méthodes qui peuvent être évoquées dans la réponse à un message d'un objet d'une classe.}

La combinaison de deux classes d'objet a comme conséquence une autre classe dont les propriétés sont l'union des propriétés des classes composantes.

Soient deux objets : X=(x, p(x)) et Y=(y, p(y))

X+Y est définie comme (z, p(z)) où z représente X+Y et p(z)= p(x) □ p(y).

A.2.4.2 Critères d'évaluation des métriques

Afin de concevoir des métriques rigoureuses, il était indispensable de concevoir une liste de critères formels, que les métriques doivent satisfaire. Weyuker (1988) a proposé un ensemble de neuf (9) propriétés pouvant être utilisées dans l'évaluation de métriques. Malgré les critiques de chercheurs comme Cherniavsky et Smith (1997) qui trouvent que « *cette liste de propriétés fournit des conditions nécessaires mais pas suffisantes pour juger de la qualité d'une métrique de complexité* », les auteurs de cet article ont décidé de l'utiliser, en apportant les modifications suivantes afin de l'adapter à l'approche OO :

1. La deuxième propriété, « granularité », est vérifiée par n'importe quelle métrique appliquée au niveau d'une classe. Puisqu'on traite un ensemble fini d'applications contenant un nombre fini de classes, donc, les classes ayant une valeur de métrique

similaire sera certainement petit. Donc, ce critère sera supprimé de la liste des critères de Weyuker (1988).

2. La huitième propriété, « propriété de renommation », sera aussi supprimée, et cela car cette dernière exige que lorsqu'une entité mesurée change de nom, sa valeur de métrique reste inchangeable, et les métriques proposées dans cet article ne dépendent pas des noms des classes, des méthodes ou encore des variables d'instance.

3. La septième propriété concerne l'impact de l'ordre des instructions dans un programme, qui une fois changé, change la valeur de la métrique. Mais, dans la Conception Orientée Objet (COO), 'ordre des instructions dans une classe n'a aucun effet sur l'utilisation ou l'exécution de la classe.

4. Les six autres propriétés de Weyuker (1988) restent intactes, en remplaçant le mot programme par classe. Une analyse de l'article dans lequel il les présente est trouvable dans l'Appendice C.

Les auteurs font les hypothèses suivantes :

Hypothèse 1 : Soient :

Xi : le nombre de méthodes d'une classe donnée i.

Yi : le nombre de méthodes appelées par une méthode donnée i.

Zi : le nombre de variables d'instance dans une classe i.

Ci : le nombre de couplages entre une classe d'objets i et les autres classes.

Xi, Yi, Zi et Ci, sont des variables aléatoires discrètes chacune caractérisée par une certaine fonction de distribution générale. Les Xi sont indépendantes et identiquement distribuées, de même pour les Yi, les Zi et les Ci. Donc, les méthodes et les variables d'instance d'une classe donnée ne peuvent pas être déduites si on connait les méthodes, les variables d'instance et les couplages des autres classes du système.

Hypothèse 2 : En général, deux classes peuvent avoir un nombre fini de méthodes « identiques » dans le sens que la combinaison des ces deux classes, est une classe dans laquelle les méthodes identiques de ces deux classes seront redondantes.

Hypothèse 3 : L'arbre d'héritage est « plein[72] », c.-à-d., il y a une racine, des nœuds intermédiaires et des feuilles.

A.2.4.3 Collecte de données empiriques

Les nouvelles métriques proposées par les auteurs ont été recueillies via des outils automatisés développés pour cette recherche en deux organismes différents : Site A et Site B.

Site A est un fournisseur de logiciel qui emploie COO dans son développement et a une collection de différentes bibliothèques de classe C++ réutilisées à travers plusieurs applications.

Des données des métriques de 634 classes de deux de ces bibliothèques qui sont employées dans la conception des interfaces utilisateur graphiques-genre prototypage et conception rapide des fenêtres.

Site B est un fabricant de semi-conducteur qui emploie le langage de programmation *Smalltalk* pour développer des systèmes de contrôle flexibles et des systèmes de fabrication. Les auteurs sont arrivés à rassembler des données des métriques de 1459 classes utilisées dans la mise en oeuvre d'un système assisté par ordinateur pour la production des circuits VLSI.

A.2.4.4 Résultats

Définitions de l'ensemble des métriques :

Méthodes pesées par lasse (Weighted Methods Per Class WMC) : On considère une classe C1 avec les méthodes M1...Mn. Si c1... cn soient les complexités de ces méthodes. Alors :

---

[72] « *Cette hypothèse déclare simplement qu'une application ne consiste pas seulement en des classes autonomes ; c'est-à-dire il y a des sous-classes* ».

$$WMC = \sum_{i=1}^{n} c_i$$

Profondeur d'arbre d'héritage (Depth of Inheritance Tree DIT): La profondeur d'héritage de la classe est la métrique DIT de la classe. Dans des cas incluant un héritage multiple, le DIT sera la longueur maximale d'un sommet de l'arbre à sa racine.

Nombre d'enfants (Number Of Children NOC) : C'est le nombre de sous-classes directement subordonnées.

Couplage entre les classes (Coupling between object classes CBO) : Le CBO d'une classe est le nombre des autres classes auxquelles elle est couplée.

Réponse pour une classe (Response For a Class RFC) : Le RFC = |RC| qui est l'ensemble de réponses d'une classe.

RS = {M} □ {Ri}

Avec :

- {Ri} : l'ensemble de méthodes appelées par la méthode i, et
- {M} : l'ensemble de toutes les méthodes de la classe.

Manque de cohésion dans les méthodes (Lack of Cohesion in Methods LCOM) : On considère une classe C1 avec les méthodes M1, M2, ..., Mn. Et {Ij} = ensemble de variables d'instance utilisées par la méthode Mi.

Il y a n ensembles {I1}...{In}. Soient P={(Ii, Ij)} (tel que l'intersection de Ii et Ij est vide) et Q={(Ii, Ij)} (tel que l'intersection de Ii et Ij est vide). Si tous les ensembles de n : {I1}...,{In} sont égaux à 0, alors P = 0.

LCOM = |P| - |Q| si |P| > |Q| , 0 autrement

Les auteurs ont procédé à l'évaluation de ces métriques selon les 6 propriétés de Weyuker (1988) qui conviennent à l'approche OO. Le tableau ci-dessous résume les résultats de cette évaluation.

Tableau A.15 Évaluation analytique [73]des métriques proposées selon les 6 propriétés de Weyuker retenues

Métrique	Propriété 1	Propriété 2	Propriété 3	Propriété 4	Propriété 5	Propriété 6
WMC	1	1	1	1	1	0
DIT	1	1	1	Il y a trois cas : 1er cas : P et Q des enfants du même parent : (1) 2ème cas :P et Q ne sont ni enfants, ni enfants du même parent : (1) 3ème cas : P est l'enfant de Q ou le contraire : (0)	1	0
NOC	1	1	1	1	1	0
CBO	1	1	1	1	1	0
RFC	1	1	1	1	1	0
LCOM	1	1	1	0	1	0

[73]Le chiffre 1 signifie que la propriété de la colonne est vérifiée par la métrique, et le chiffre 0 signifie le contraire.

On remarque que toutes les métriques satisfont la majorité des propriétés, sauf la sixième propriété qui n'est satisfaite par aucune de ces métriques, cette propriété implique que lorsqu'une classe est subdivisée en deux classes, sa complexité diminue.

$$C(P) + C(Q) < C(P + Q)$$

Mais, les concepteurs qui ont participé dans cette étude ont remarqué le contraire, parce que la gestion de la mémoire et la détection d'erreur deviennent plus compliquées avec un nombre élevé de classes.

Les auteurs concluent que cette propriété ne sert pratiquement à rien dans l'évaluation des métriques de complexité applicables sur des projets OO.

Ils font aussi remarquer que les métriques DIT et LCOM, ne vérifient pas toujours la quatrième propriété. LCOM sous certaines conditions ne la vérifie pas, alors que DIT, dans le cas où les classes ont un lien de parenté, ne la vérifie pas. Cela est dû au fait que la distance de la racine au parent ne peut pas devenir plus grande que celle de ses descendants.

La Figure A.20 et la Figure A.21, représentent les histogrammes de la métrique WMC des deux sites A et B respectivement. À première vue, on remarque la similitude de la distribution des valeurs de la métrique dans les sites A et B, malgré leurs différences. On constate aussi que la majorité des classes de ces deux applications « *ont des constructions simples et fournissent une abstraction et une fonctionnalité uniques* ».

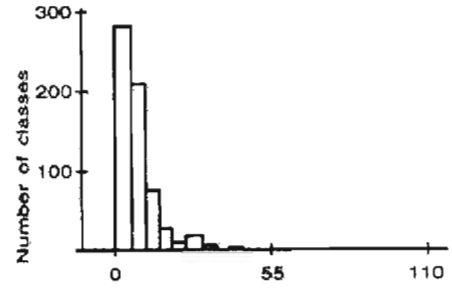


Figure A.20 Histogramme de la métrique WMC du Site A

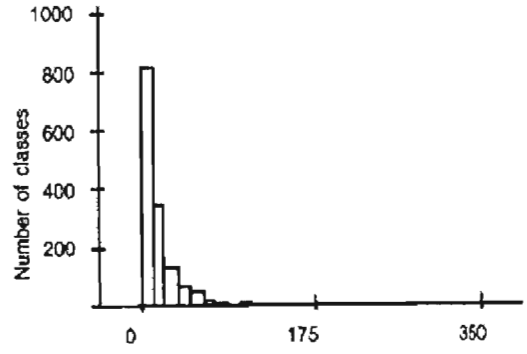


Figure A.21 Histogramme de la métrique WMC du Site B

La Figure A.22 et la Figure A.23, illustrent les histogrammes de la métrique DIT des sites A et B respectivement. On remarque qu'au niveau des deux sites la bibliothèque semble avoir beaucoup de classes proches de la racine, plutôt que beaucoup de classes proches du fond de la hiérarchie. Les tests deviennent une tâche difficile avec des classes d'héritage très élevé, comme le cas d'une classe du Site A, qui a uniquement 4 méthodes et des variables locales, mais ses objets ont accès à 132 méthodes via l'héritage, ce qui rend cette classe compliquée à tester.

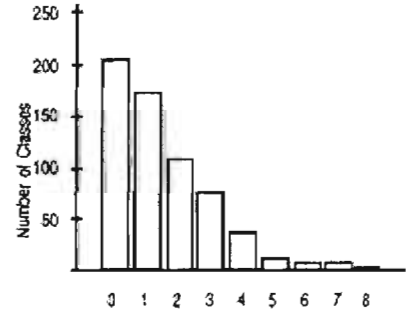


Figure A.22 Histogramme de la métrique DIT du Site A

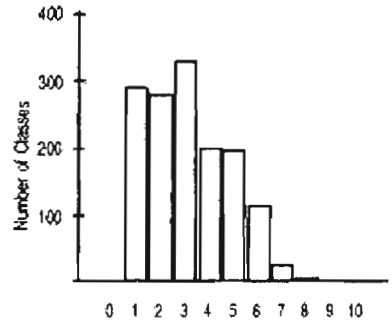


Figure A.23 Histogramme de la métrique DIT du Site B

À partir des histogrammes de la métrique NOC des Sites A et B présentés respectivement dans la Figure A.24 et la Figure A.25, les auteurs remarquent qu'il y a une similitude dans la nature de distribution des valeurs de cette métrique dans les deux sites. Sous cet aspect elle ressemble à la métrique WMC. En général, on remarque que peu de classes ont des enfants directs, à peu prés 73 % des classes du site A et 68 % du site B, n'ont pas d'enfants. Donc on peut déduire que l'héritage n'est pas vraiment une technique de base dans la conception des classes, et cela peut être dû, au manque de communication entre les concepteurs. « *L'utilisation systématique de cette métrique, peut aider à restructurer la hiérarchie des classes afin d'exploiter les caractéristiques communes* ».

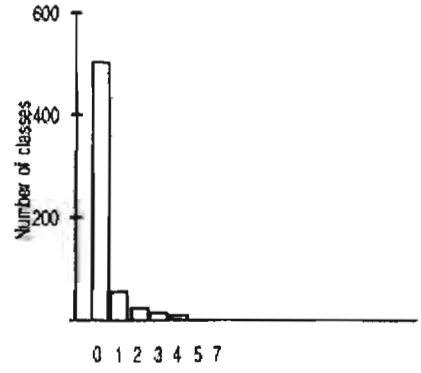


Figure A.24 Histogramme de la métrique NOC du Site A

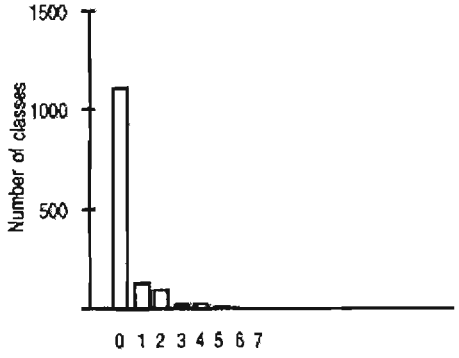


Figure A.25 Histogramme de la métrique NOC du Site B

On observe une assez grande différence entre les valeurs de la métrique CBO des sites A et B dans la Figure A.26 et la Figure A.27. Cela à cause de la nature du langage *Smalltalk* utilisé dans le site B, « *qui exige pratiquement que chaque interaction entre les entités d'exécution s'établit par un envoi de messages, et les variables scalaires simples (nombres entiers, reals, et caractères) et les instructions de construction de flux de contrôle comme : If, While, repeat, sont considérés des objets* ». Alors que ce n'est pas le cas pour le langage C++ utilisé dans le site A. Et d'autre part, le site B contient beaucoup plus de classes (1459 classes) comparé au site A (634), donc il serait normal que les valeurs de la métrique CBO dans le site B soient supérieures à celles dans le Site A, car « *le couplage entre les classes est une fonction qui s'incrémente avec le nombre de classes dans une application* ».

Enfin, la métrique CBO peut être utilisée par les concepteurs et les chefs de projets, comme « *manière simple et relative pour savoir si la hiérarchie de classe perd son intégrité, et si les différentes parties d'un grand système développent des interconnexions inutiles dans des endroits inadéquats* ».

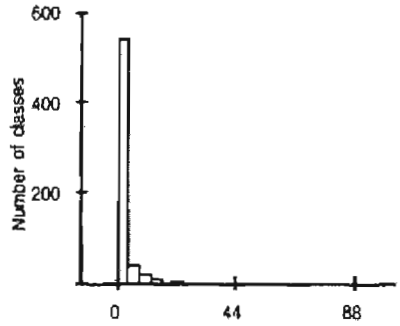


Figure A.26 Histogramme de la métrique CBO su Site A

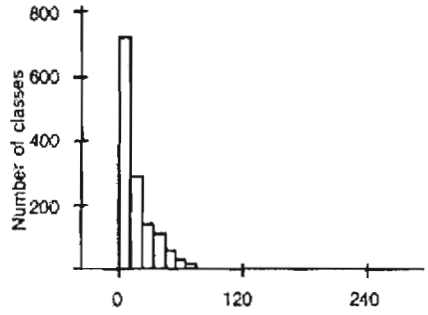


Figure A.27 Histogramme de la métrique CBO du Site B

Les histogrammes de la métrique RFC des sites A et B sont présentés par la Figure A.28 et La Figure A.29. On remarque que généralement la majorité des classes tendent à appeler un petit nombre de méthodes. Et comme le cas de la métrique CBO, on remarque aussi que les valeurs de la métrique RFC du site B sont largement plus grandes que celles du site A. Et cela est toujours dû aux principes OO de *Smalltalk*.

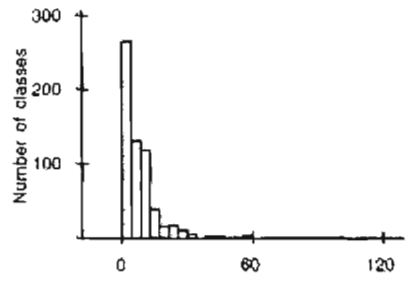


Figure A.28 Histogramme de la métrique RFC du Site A

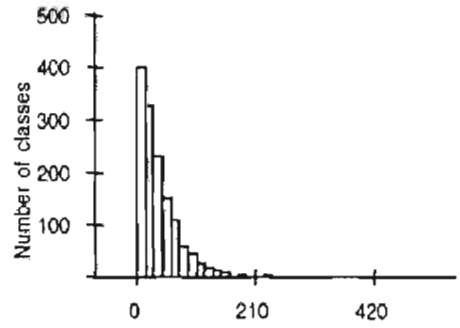


Figure A.29 Histogramme de la métrique RFC du Site B

La Figure A.30 et la figure A.31 ci-dessous montrent bien que, au niveau des sites A et B, au moins 50 % des classes comprennent des méthodes cohésives, avec une différence de la forme de distribution de LCOM bien claire. « *Une valeur élevée de LCOM indique qu'il ya une dispersion dans la fonctionnalité fournie par la classe. Cette métrique peut être employée pour identifier les classes qui essayent d'atteindre beaucoup d'objectifs, et qui sont susceptibles de se comporter de manières moins prévisibles que les classes qui ont des valeurs plus basses de LCOM* ».

Cette métrique peut être un moyen pour vérifier si les principes de cohésion ont été bien pris en compte dans la conception et aussi un moyen pour proposer des changements si elle est utilisée dans les premières phases du cycle de conception.

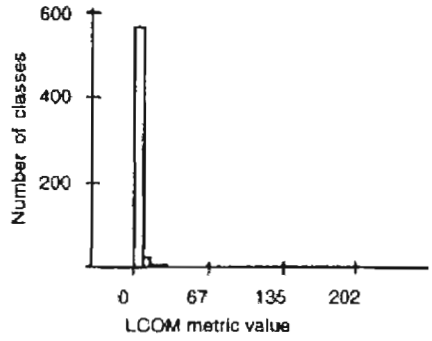


Figure A.30 Histogramme de la métrique LCOM du Site A

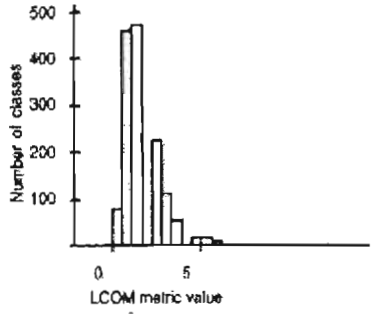


Figure A.31 Histogramme de la métrique LCOM du Site B

Récapitulatif :

Le tableau ci-dessous représente les étapes de Booch (1991) pendant lesquelles les nouvelles métriques peuvent être utilisées. Les auteurs décrivent cette utilisation comme suit :

- Les métriques WMC, DIT et NOC, sont liées à l'étape identification des classes, puisque WMC est un aspect de la complexité de la classe et DIT et NOC sont reliées directement à la disposition de la hiérarchie de la classe.

- WMC et RFC capturent le comportement des objets d'une classe quand ils reçoivent des messages, et la métrique LCOM est liée à l'empaquetage des données et les méthodes dans une définition de classe, et mesure la cohésion de cette dernière..

- Les métriques RFC et CBO capturent également l'ampleur de communication entre les classes en comptant les couples inter-classes et les méthodes externes d'une classe donnée, de ce fait elles sont liées à la troisième étape (Identification des relations entre les classes) dans la COO de Booch.

Tableau A.16 Traçage des métriques aux étapes de COO de Booch

Metric	Identification	Semantics	Relationships
WMC	X	X	
DIT	X		
NOC	X		
RFC		X	X
CBO			X
LCOM		X	

A.2.4.5 Directions futures :

Cette suite de métriques a commencé à être utilisée par certains organismes. En effet, elle a été employée par les services d'informatique de Boeing, dans l'évaluation de différentes méthodologies OO. Et récemment Li et Henry ont utilisé ce groupe de métriques dans l'étude de la variance de l'effort de la maintenance, ils ont remarqué que cette suite de métriques donne plus d'explications que les métriques traditionnelles concernant la taille.

Les travaux futurs suggérés par les auteurs sont :

- Étude de la capacité de ces métriques dans l'établissement préliminaire de l'évaluation des langages de programmation et des environnements.
- « *Analyse du degré de corrélation de ces métriques et les indicateurs de performance, comme, l'effort de la conception, les tests et la maintenance, la qualité et la performance du système* ».
- Ces métriques peuvent guider la gestion de la complexité d'une application donnée, en fournissant une idée sur l'évolution de la complexité avec le développement de l'application, et cela en appliquant ces métriques sur un projet commercial de la phase conception jusqu'au déploiement et aux diverses étapes intermédiaires.

A.2.4.6 Références

Les auteurs ont utilisé cinquante références:

- Dix livres, trois qui datent des années 1970, deux des années 1980 et cinq des années 1990.
- Trente-sept articles :
    - Trois des années 1970.
    - Neuf des années 1980, parmi eux l'article de Weyuker (1988) que nous avons résumé dans l'Appendice C.
    - Vint-cinq des années 1990.
- Un rapport technique qui date des années 1990.
- Un mémoire de maîtrise
- Une référence dont nous ne connaissons pas la nature

A.2.5 Bieman et Kang (1998) « Mesure de la cohésion au niveau de la conception »

Comme pratiquement tous les auteurs qui traitent de la cohésion, les auteurs de cet article partent de la définition de cohésion de Yourdon et Constantine (1979) que nous avons analysée dans le chapitre III. En ce qui concerne les méthodes de mesure, ils s'appuient sur deux approches définies par d'autres auteurs :

- L'approche fondée sur les associations de Lakhotia (1993) que nous avons analysée dans la section A.2.1.
- L'approche fondée sur le tranchage de Bieman et Ott (1994).

A.2.5.1 Mesures fondées sur les associations.

Les auteurs définissent six (6) types de relations pouvant exister entre une paire de composants[74], et qui correspondent[75] aux six (6) niveaux d'association de Yourdon et Constantine (1979).

Avant de définir ces relations, on présente dans le tableau ci-dessous les différents types de dépendances entre une paire de composants, comme ils ont été présentés par les auteurs.

Tableau A.17 Types de dépendances entre paires de composants

Le type de dépendance	Définition de la dépendance	Représentation
Dépendance de données	**y**[76] a une dépendance de données à l'égard de **x**, si **x** accède à **y** à travers un chemin consistant en une chaîne de définitions.	d (x —>y)
Dépendance de contrôle	**y** a une dépendance de contrôle à l'égard de **x**, si la valeur de **x** détermine si l'instruction contenant **y** sera exécutée.	
Dépendance	**y** dépend de **x**, quand il y a un chemin de **x** à **y**, à travers une séquence de dépendance de données ou de contrôle. Ce chemin est nommé chemin de dépendance.	(x —>y)
Dépendance contrôle conditionnel	**y** a une dépendance de contrôle conditionnel à l'égard de **x**, si **y** a une dépendance de contrôle à l'égard de **x**, et **x** est utilisée dans le prédicat d'une structure de décision.	cc (x —>y)
Dépendance contrôle de type itération	**y** a une dépendance de contrôle de type itération à l'égard de **x**, si **y** a une dépendance de contrôle à l'égard de **x**, et **x** est employé dans le prédicat d'une	ic (x —>y)

[74] Selon l'article, « *les composants d'entrée d'un module incluent les paramètres d'entrée et les variables globales référencées. Les composants de sortie incluent les paramètres de sortie, les variables globales modifiées et les valeurs retournées par les fonctions* ».
[75] La cohésion temporelle n'est pas incluse.
[76] X et Y sont deux variables.

Le type de dépendance	Définition de la dépendance	Représentation
	structure d'itération.	
Dépendance contrôle-c	**y** a une dépendance de type contrôle-c à l'égard de **x**, si le chemin de dépendance entre **x** et **y** contient une dépendance de contrôle-condition mais pas de dépendance de contrôle-itération.	c (x —->y)
Dépendance contrôle-i	**y** a une dépendance de type i-contrôle à l'égard de **x**, si le chemin de dépendance entre **x** et **y** contient une dépendance de contrôle-itération.	i (x —>y)

Ces relations sont basées sur la représentation du graphe de dépendance des entrées-sorties (*Input/Output Dependance Graph* IODG), qui n'est autre que le graphe de dépendance des variables de Lakhotia (1993) adapté, et qui est basé sur les relations de dépendance des données et des contrôles entre les composants d'entrée-sortie d'un module. Mathématiquement défini comme un graphe dirigé GM = (V, E), où V est l'ensemble de composants d'entrée-sortie de M, et E l'ensemble d'arêtes marquées avec les types de dépendance, tels que E = {(x, y) □ V × V | y a une dépendance de données, de contrôle-c, et/contrôle-i sur x.}

On définit ces relations comme suit :

1) Relation accidentelle (*R*1):

$$R_1(o_1, o_2) = o_1 \neq o_2 \wedge \neg(o_1 \rightarrow o_2) \wedge \neg(o_2 \rightarrow o_1) \wedge$$
$$\neg\exists x\,[(x \rightarrow o_1) \wedge (x \rightarrow o_2)]$$

Les deux sorties o1 et o2 du module ne dépendent pas l'une de l'autre, et ne dépendent pas de la même entrée.

2) Relation conditionnelle (*R*2):

$$R_2(o_1, o_2) = o_1 \neq o_2 \wedge \exists x\, [(x \xrightarrow{c} o_1)] \wedge (x \xrightarrow{c} o_2)]$$

Les deux sorties ont une dépendance de type contrôle-c d'une entrée commune.

3) Relation itérative (*R*3):

$$R_3(o_1, o_2) = o_1 \neq o_2 \wedge \exists x\, [(x \xrightarrow{i} o_1) \wedge (x \xrightarrow{i} o_2)]$$

Les deux sorties ont une dépendance de type contrôle-i d'une entrée commune.

4) Relation communicationnelle (*R*4):

$$R_4(o_1, o_2) = o_1 \neq o_2 \wedge \exists x\, [((x \xrightarrow{a} o_1) \wedge (x \xrightarrow{a} o_2) \vee ((x \xrightarrow{p} o_1) \wedge (x \xrightarrow{q} o_2))], \text{ where } p, q \in \{d, c, i\}, \text{ and } p \neq q.$$

Les deux sorties dépendent de la même entrée, ou une entrée est employée pour calculer les deux sorties, mais ni comme un drapeau de condition pour choisir l'une des deux sorties ni comme une boucle invariable pour les calculer.

5) Relation séquentielle (*R*5):

$$R_5(o_1,o_2) = o_1 \neq o_2 \wedge ((o_1 \rightarrow o_2) \vee (o_2 \rightarrow o_1))$$

Une sortie dépend de l'autre.

6) Relation fonctionnelle (*R*6):

$$R_6(o_1,o_2) = (o_1 = o_2)$$

Il y a une seule sortie dans tout le module.

Les auteurs définissent la première mesure basée sur l'approche association entre les éléments de traitement du module comme suit :

« Mesure de cohésion au niveau de la conception (Design-Level Cohesion DLC*): Le niveau de cohésion d'un module est déterminé en fonction des niveaux de relation entre les différentes paires de sorties. La relation d'une paire de sortie, est la relation la plus forte parmi toutes ses relations. Le niveau de cohésion du module est le plus faible (le niveau le plus bas) de toutes ses paires de sorties. Ce qui veut dire que, la cohésion du module correspond au niveau le plus faible entre toutes ses paires* ».

La figure ci-dessous, représente six modules avec six niveaux de cohésion différents, avec leurs diagrammes de graphe de dépendance des entrées/sorties.

Les auteurs mesurent la cohésion fonctionnelle en termes de connexions entre les jetons de données du code dans des tranches de sorties du module.

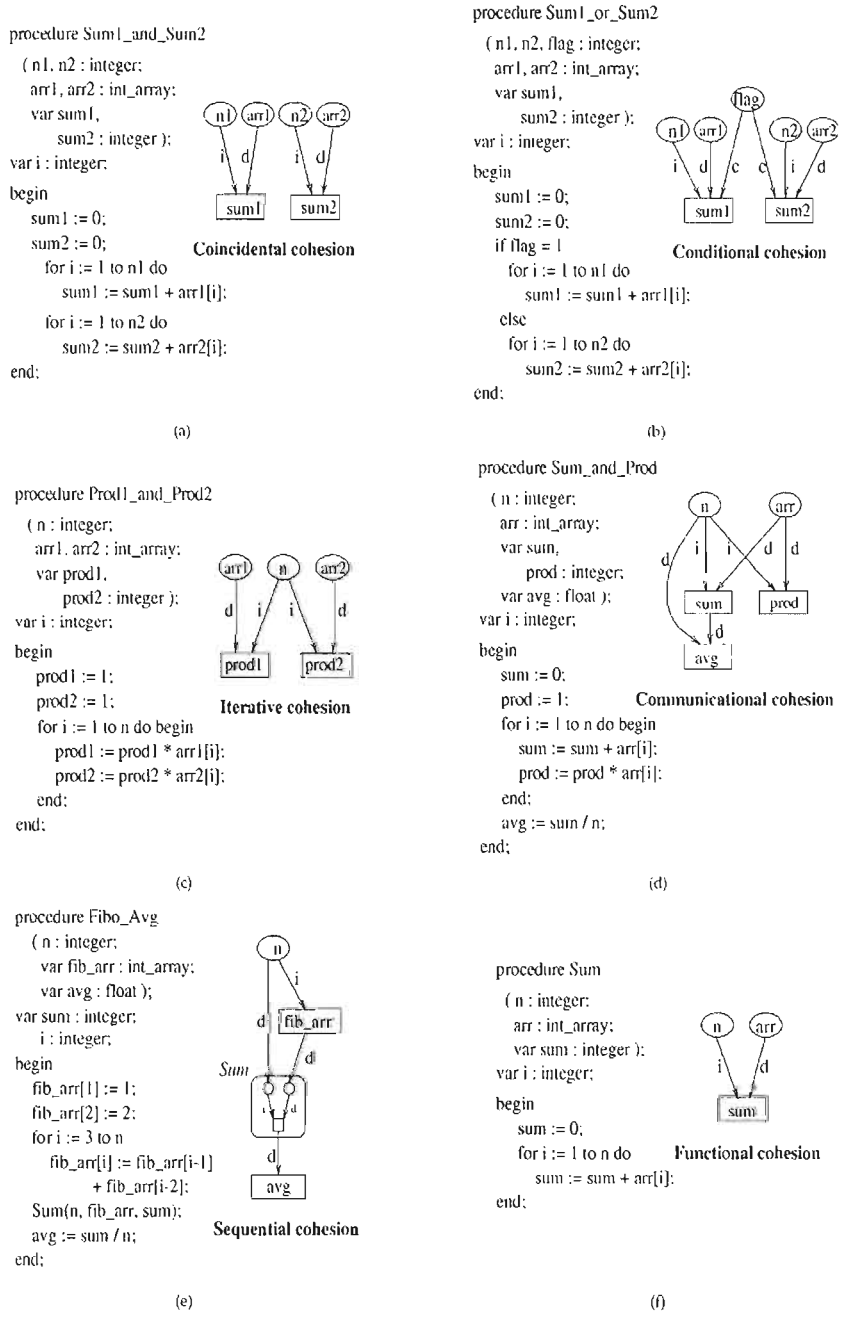


Figure A.32 Exemple de modules avec graphes de dépendance et avec niveaux de DLC

A.2.5.2 Mesures fondées sur le tranchage

Pour la deuxième mesure, les auteurs se basent sur l'approche de *tranchage*, employée par Bieman et Ott (1994) dans la définition de trois mesures de la cohésion fonctionnelle (*Functional Cohesion FC*) —la cohésion fonctionnelle faible (WFC), la cohésion fonctionnelle forte (SFC) et l'adhérence (A) des *tranches de données*[77]. » C'est à partir de ces dernières que les auteurs ont dérivé de nouvelles mesures, appelées mesures de cohésion fonctionnelle au niveau de la conception (*Design-level Functional Cohesion* DFC[78]), qui sont appliquées sur des modèles simplifiés d'IODG, comprenant seulement les relations de dépendance entre les différentes entrées/sorties du module. Ces mesures (les mesures de DFC) sont exprimées en termes de nombre de composants isolés, nombre de composants essentiels et de connexité des composants, que l'on définit comme suit :

- ***composant isolé*** : est un composant qui affecte seulement une fonctionnalité locale, c.-à-d., il a une relation de dépendance avec seulement une sortie.

- ***composant essentiel*** : est un composant qui affecte (ou est affecté par) toutes les fonctionnalités du module, c.-à-d., il a des relations de dépendance avec toutes les sorties du module.

- ***connexité d'un composant*** : est le degré de parenté (*relatedness*) du composant par rapport aux sorties.

La connexité de l'$i^{ème}$ composant d'un module donné, est :

$$C_i = \begin{cases} \frac{N_i - 1}{O - 1} & \text{if } O > 1 \\ 1 & \text{otherwise} \end{cases}$$

---

[77] Une tranche de données d'une variable, est définie dans l'article comme, « *la séquence de jetons de données qui dépendent de la variable* ».

[78] Les mesures de DFC sont dérivées des mesures de FC, dans la mesure où toutes les deux sont définies en termes de connexions entre les composants et les sorties, sauf que les composants pour les mesures de DFC incluent uniquement les entrées/sorties, vu que ce sont les seuls composants visibles lors de la conception, contrairement aux mesures de FC appropriées à la phase codage.

Avec Ni, le nombre de sorties avec lesquelles l'i^{ème} composant a une relation de dépendance et O, le nombre total de sorties dans le modèle d'IODG du module.

Les auteurs définissent trois (3) mesures de DFC :

4. ***Cohésion faible*** (*Loose Cohesiveness LC*) : nombre relatif de composants non isolés :

$$\mathrm{LC}(m) = \mathrm{D}/\mathrm{T}$$

5. ***Cohésion forte*** (*Tight Cohesiveness TC*) : nombre relatif de composants essentiels :

$$\mathrm{TC}(m) = \mathrm{E}/\mathrm{T}$$

6. ***Cohésion de module*** (*Module Cohesiveness MC*) : connexité moyenne des composants du modèle :

$$\mathrm{MC}(m) = \frac{\sum_{i=1}^{T} C_i}{T}$$

Avec :

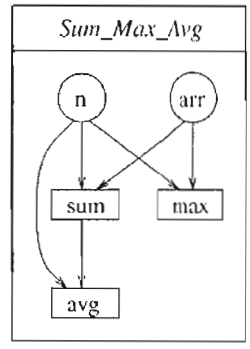
- M : le module.
- D : le nombre de composants non isolés.
- E : le nombre de composants essentiels.
- *Ci* : la connexité du ième composant.
- T : le nombre total de composants dans le module.

Les auteurs présentent un exemple de programme Sum_Max_Avg, avec son profil de tranches, son IODG et sa table des dépendances des entrées/sorties (*Input/Ouput Dependance*

Table IODT). Dans la première ligne de la table, on trouve les noms des sorties, et dans la première colonne on trouve les noms des composants (les entrées/sorties). Le chiffre 1 dans une case veut dire qu'il y a une dépendance entre le composant et la sortie correspondante à cette case. Et le chiffre 0 montre le contraire. Cet exemple est présenté dans le tableau ci-dessous.

Tableau A.18 Exemple de programme avec son profil de tranche et son IODG

```
procedure Sum_Max_Avg
 ( n : integer; /* pre: n>0 */
   var arr: int_array;
   var sum,
       max : integer;
    var avg : float );
   i : integer;
begin
   sum := 0;
   max := arr[1];
   for i := 1 to n do begin
      sum := sum + arr[i];
      if arr[i] > max
         max := arr[i];
   end;
   avg := sum / n;
end;
```



component \ output	sum	max	avg
n	1	1	1
arr	1	1	1
sum	1	0	1
max	0	1	0
avg	1	0	1

En analysant l'IODG et l'IODT du programme de l'exemple, nous construisons le tableau suivant dans lequel nous présentons les conclusions tirées de cette analyse.

Tableau A.19 Conclusion tirées de l'exemple du Tableau A.18

Composant	Connexité calculée	Type du composant	Commentaire
n	1	Essentiel	Le composant « n » a des relations d'indépendance avec toutes les sorties : sa connexité est égale à 1, donc c'est un composant essentiel.
arr	1	Essentiel	Idem.
sum	1/2	Non-isolé	Csum = (2-1)/ (3-1) = 1/2
max	0	Isolé	Le composant « max » n'a qu'une seule relation de dépendance qui est avec lui-même, donc c'est un composant isolé.
avg	1/2	Non-isolé	Cavg = (2-1)/ (3-1) = 1/2

En appliquant les mesures de DFC, toujours sur le programme de l'exemple, on aura comme résultats :

LC (Sum_Max-Avg) = 4/5 = 0.8

TC (Sum_Max-Avg) = 2/5 = 0.4

MC (Sum_Max-Avg) = (1+1+1/2+1/2)/5 = 0.6

Afin de valider ces nouvelles métriques, les auteurs font une étude comparative avec la métrique FC. Tout d'abord, ils se fondent sur une comparaison analytique pour déceler les convergences et les divergences entre ces métriques.

Le tableau ci-dessous résume les résultats de cette comparaison.

Tableau A.20 Comparaison entre les nouvelles métriques et la métrique FC

Couple de métriques	Points de ressemblance	Points de dissemblance	Déductions
DFC vs FC	Les deux métriques mesurent le degré de dépendance entre les composants du module.	Différence dans la définition de ces deux mesures, FC est définie en termes de relations entre les composants de l'interface du module, et DFC en termes de relations entre ceux du corps du module. Les mesures de FC fournissent plus d'informations détaillées sur la restructuration des modules que les mesures de DFC. Les mesures de DFC décèlent mieux les relations entre les entrées et sorties.[79]	Il y a généralement une certaine correspondance entre ces mesures : * Si FC = 1 alors DFC =1 * Si FC = 0 alors DFC = 0 * Si 0 < DFC < 1 alors : - Quand FC > DFC, cela veut dire que le module mesuré contient beaucoup plus de données essentielles que de composants d'entrées/sorties essentiels. - Quand DFC > FC, cela veut dire que le module mesuré comprend beaucoup plus de données isolées que des composants d'entrée/sortie isolés.
DLC[80] vs DFC	*« Les deux ensembles de mesures ont été définis en utilisant une compréhension intuitive de la cohésion basée sur la relation des composants du module ».*	Les auteurs étudient les différentes divergences entre l'ensemble de mesures, en étudiant l'effet de deux variables pouvant les influencer.	On en déduit de cette étude les préceptes suivants : * Les résultats des mesures DFC et DLC, doivent être interprétées différemment[81]. * Le nombre de composants (essentiels et isolés) a une grande influence sur les mesures DFC, tandis qu'il ne produit aucun effet sur la DLC. * DLC fournit des « *informations plus précises* » caractérisant les relations entre les composants de sortie que celles fournies par DFC. * La mesure TC [82] de l'ensemble des mesures DFC, est la mesure qui correspond[83] le plus à la DLC.

[79] Et cela parce que les mesures de FC sont au fond basées sur le détail interne du module comme déjà mentionné.
[80] Pour comparer la mesure DLC aux trois mesures de DFC, l'auteur utilise l'IODG simplifié. Avec ce dernier la différence entre les degrés de cohésion : conditionnelle, itérative et communicationnelle ne peut pas être décelée, donc on leur donne tous le même nom qui est « *cohésion indirecte* ».
[81] Cela est bien évident, puisque l'information fournie par ces mesures n'est pas du tout la même, puisque la DLC détecte la plus faible connexion dans le module, alors que l'ensemble des mesures DFC, nous procure la moyenne parmi toutes ces connexions.
[82] À rappeler que la TC, capture le nombre de composants essentiels de l'interface du module.
[83] « *DLC et TC sont calculées en utilisant les cas les plus extrêmes* ».

Après la comparaison analytique résumée dans le tableau précédent, les auteurs présentent sous forme de tableau les valeurs numériques pour une comparaison empirique, les Tableau A.21 et le Tableau A.22.

Tableau A.21 Moyenne et médiane des mesures FC, DFC et DLC pour 607 fonctions C

Average	No. of data tokens	No. of input/outputs	FC			DFC			DLC
			WFC	ADH	SFC	LC	MC	TC	
Mean	57.36	6.35	0.75	0.72	0.68	0.78	0.74	0.71	4.76
Median	30.00	4	1	1	1	1	1	1	1

Tableau A.22 Moyenne et médiane des mesures FC, DFC et DLC pour 264 fonctions C apr`s avoir enlevé les fonctions avec une seule sortie

Average	No. of data tokens	No. of input/outputs	FC			DFC			DLC
			WFC	ADH	SFC	LC	MC	TC	
Mean	83.77	8.98	0.42	0.36	0.27	0.50	0.41	0.33	3.16
Median	52	8	0.42	0.31	0.14	0.50	0.33	0.20	4

Tableau A.23 Comparaison empirique détaillée des trois mesures FC, DFC et DLC

Relations entre FC & DFC	Relations entre DFC & DLC	Relations entre FC & DLC
* FC et DFC sont fortement corrélées, et cela est justifié par la corrélation forte existante entre chaque paire de mesures correspondantes dans les deux ensembles de mesure. Cela emmène l'auteur à confirmer que « *le degré de cohésion d'un programme peut être prévu lors de la phase conception.* » * On remarque, que les mesures DFC sont notablement plus grandes que celles de FC, comme l'a montré l'étude empirique (on voit deux droites plus ou moins parallèles, ce qui appuie l'idée soulevée auparavant.) * La corrélation dont on a déjà parlé est affectée par la taille du programme, du fait qu'elle est plus grande pour les petits programmes que pour les plus grands. * Cette corrélation est aussi affectée par le nombre d'annexes, plus ce nombre est petit plus la corrélation est importante.[84]	Les résultats de cette comparaison correspondent bel et bien aux résultats obtenus de l'étude comparative analytique effectuée auparavant, c.à.d.: * Il existe une corrélation entre les mesures DFC et DLC. * Parmi les valeurs de l'ensemble des mesures de DFC, il y a celles de TC qui sont les plus proches des valeurs de DLC. * Les mesures DFC, sont sensibles au nombre de composants et au nombre de connexions entre ces derniers, alors que DLC ne prouve aucune sensibilité de ce genre. Et cela, on le voit à travers la forte corrélation qui existe entre ces mesures quand il s'agit de petites interfaces.	*« Nous constatons que le rapport entre FC et DLC est très semblable à celui entre DFC et DLC sauf que la corrélation entre FC et DLC est plus faible que celle entre les mesures de DFC et de DLC ».* **Coefficients de corrélation** WFC-DLC : 0.6007 ADH-DLC : 0.7035 SFC-DLC : 0.8202

[84] Les auteurs ont remarqué cela en comparant les taux de corrélation des mesures DFC et FC pour les programmes système et les programmes des étudiants, cette comparaison a montré que les DFC et FC des programmes système sont plus corrélés que ceux des étudiants.

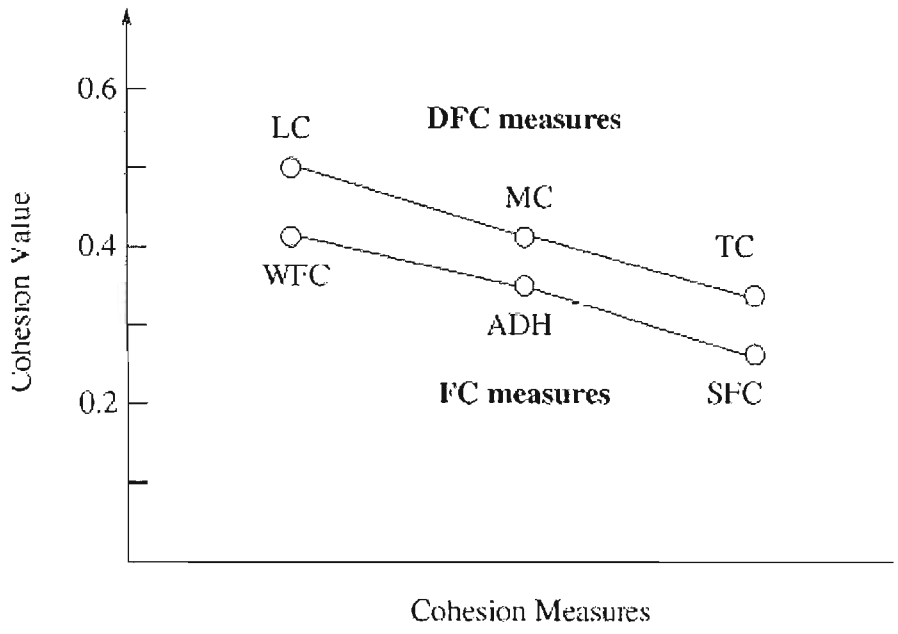


Figure A.33 Comparaison empirique entre l'Ensemble des métriques de DFC et FC

Tableau A.24 Coefficients de corrélation entre WFC&LC, ADH&MC et SFC&MC pour chaque partition des programmes collectés

Site/Size	No. of functions	Mean and Median data tokens		Correlation Coefficient		
				WFC-LC	ADH-MC	SFC-TC
Student Programs	141	95.36	53	0.8412	0.8437	0.8526
System Programs	123	70.47	51	0.9285	0.9418	0.9556
data token cnt ≤ 50	129	25.51	24	0.9535	0.9486	0.9565
50 < data token cnt ≤ 100	69	73.61	73	0.8286	0.8337	0.8720
100 < data token cnt	66	208.24	172	0.8065	0.8242	0.8208
Total	264	83.77	52	0.8896	0.8939	0.9043

Tableau A.25 Coefficients de corrélation de DLC&LC, DLC&MC et DLC&TC pour chque partition des programmes collétcés

Site/Size	No. of functions	Mean and Median data tokens		Correlation Coefficient		
				DLC-LC	DLC-MC	DLC-TC
Student Programs	141	8.18	6	0.6122	0.7593	0.8635
System Programs	123	9.89	8	0.6753	0.8289	0.8715
1 < I/O cnt ≤ 5	67	4.31	4	0.8751	0.9325	0.9259
5 < I/O cnt ≤ 9	112	7.11	7	0.5597	0.7385	0.8814
9 < I/O cnt	85	15.12	13	0.4845	0.6926	0.7852
Total	264	8.98	7	0.6345	0.7884	0.8760

A.2.5.3 Références

Les auteurs ont utilisé vingt (20) références :

- Trois livres, dont deux sont les livres de Yourdon et Constantine (1979) et de Stevens, Myers et Constantine (1974) et un qui date des années 1990.
- Seize articles:
    - Trois publiés en 1980, dont un est celui d'Emerson (1984) qui a été analysé et synthétisé.
    - Treize des années 1990, dont trois ont été synthétisés et analysés, il s'agit des articles de Lakhotia (1993), de Woodward (1993) et de Chidamber et Kemerer (1994).
- Une référence dont nous ne connaissons pas la nature.

A.2.6 Gall, Hajek et Jazayeri (1998) « Détection du couplage logique en se basant sur l'historique des versions du produit »

Dans cet article, les auteurs étudient le logiciel[85] de la partie commutation d'un système de télécommunication. Il s'agit d'un système complexe coûteux et difficile à

[85] Leur évaluation se rapporte seulement à la partie logicielle du système.

maintenir, « *car le produit final devra être adapté aux différents marchés et aux différentes applications, ce qui exige une personnalisation effectuée principalement par des changements qui touchent directement le code* ».

Ce logiciel comporte plus de 10 millions de lignes de code et des milliers de fichiers. Les auteurs dans leur étude ont analysé les 20 versions produites dans les deux dernières années.

La figure ci-dessous montre sous forme d'arbre de 4 niveaux la structure du logiciel, structure qui « *a été définie après la mise en œuvre de beaucoup de versions du système et qui représente la structure organisationnelle plutôt que la structure de la mise en œuvre* ».

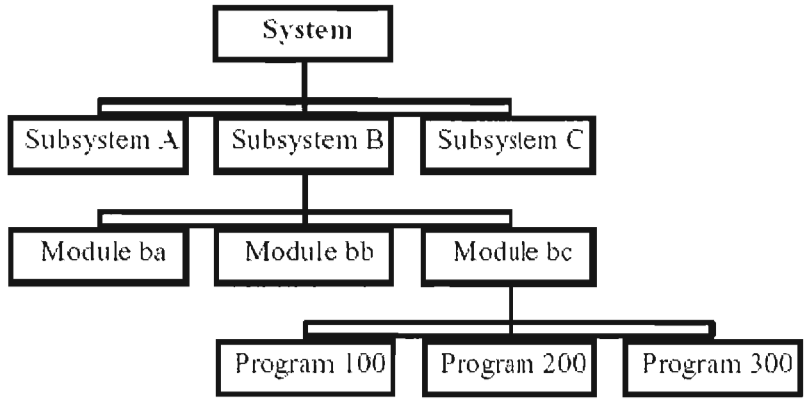


Figure A.34 Structure du logiciel

Les 20 versions du système sont stockées dans une base de données. Et chacune de ces versions, se compose de 8 sous-systèmes, de 47 à 49 modules et d'environ 1500 à 2300 programmes. Ces programmes sont codés en langage SDL[86], ensuite traduits en C, et compilés en utilisant le compilateur de C.

---

[86] Un langage populaire de haut niveau dans les systèmes de télécommunication.

Les auteurs donnent quelques définitions jugées utiles pour comprendre les représentations des programmes et leurs modèles de changement sous forme de nombres et de séquence de nombres. Les définitions telles que présentées dans l'article, sont :

- ***Une séquence de changements*** est définie au niveau des programmes et est un n-tuple <1 2..n> des nombres de versions des programme qui ont changé de numéro de version. Une séquence de changement d'un programme contient tous les numéros des versions du système dans lesquels le programme change. Par exemple dans le tableau ci-dessous, on note la séquence de changement du programme pi comme suit : <3 5> et qui dénote un changement de pi dans les versions 3 et 5 du système.

- ***Une sous-séquence (SUB)*** est une partie contiguë d'une séquence. Les changements sont représentés par une séquence ou une sous-séquence.

- ***Un rapport de changement (Change Report CR)*** est un rapport contenant le numéro de versions du programme qui a changé avec les descriptions des changements effectués.

Tableau A.26 Les numéros de versions et la représentation des séquences de changement du programme pi

System Release	1	2	3	4	5	6	7	8	9
P_i version no.			1.1	1.1	1.3	1.3	1.3	1.3	
P_i change sequence			3		5				

L'approche utilisée pour identifier les changements utilise deux processus :

1. ***Analyse de la séquence des changements (Change Sequence Analysis CSA) :*** identifie les modèles de changement. Chaque changement d'un module est lié au niveau-système. Une séquence de changement pour un module montre les versions dans lesquelles le module a été changé. À partir de ces séquences de changements, nous pouvons définir le couplage logique et cela en identifiant un ensemble de modèles communs de changement des modules donnés.

2. ***Analyse du rapport de changement (Change Report Analysis CRA) :*** Il décrit les raisons, la catégorie d'erreur, la quantité et le type d'un changement d'un programme d'une version du logiciel particulière. Si les rapports de changement identifient la même raison du changement , dans les programmes avec le même ordre de changement, alors le couplage logique identifié dans le processus de CSA est vérifié.

A.2.6.1 CSA

L'étude est faite au niveau des programmes, et cela parce que le niveau module n'est pas représentatif, puisque les modules contiennent tous les changements exécutés au niveau des programmes. Ceci a comme conséquence un nombre élevé de changements pour chaque module. Si un programme change son numéro de version, le module contenant doit également changer son numéro de version.

On considère deux types de couplage dans le processus CSA :

1. *Le couplage du système* : c'est le degré de liaison par des séquences entre les sous-systèmes. On dit que deux sous-systèmes sont couplés s'ils sont reliés à la même séquence et contiennent la même sous-séquence. Pour définir ce couplage on procède comme suit :

    a. On liste les séquences de changement de tous les programmes,

    b. Les différentes sous-séquences sont comparées.

    c. Si plus que deux séquences contiennent la même sous-séquence, on suppose qu'il existe un couplage logique entre ces séquences et bien évidemment entre les programmes et les sous-systèmes associés.

    d. Enfin pour déterminer si le couplage représente de vraies dépendances, il est nécessaire d'inspecter les rapports de changement par le processus CRA.

La figure ci-dessous est un modèle simplifié du système étudié, avec ses séquences de changements correspondantes. Les sous-systèmes sont représentés par des cercles[87], tandis

[87] Les auteurs ont changé les noms des sous-systèmes pour que le vrai système ne soit pas identifié.

que les séquences de changement sont représentées par des lignes, qui diffèrent les unes des autres et cela selon la quantité de changement contenue dans la séquence.

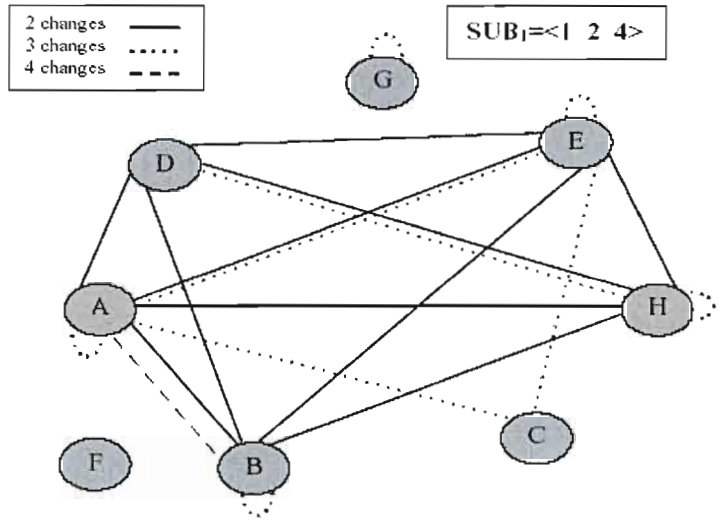


Figure A.35 Couplage entre les sous-systèmes

2. **Le couplage de séquence** : il représente les relations entre les différentes séquences via les sous-systèmes. La figure ci-dessous montre ce type de couplage dans le système étudié, les boîtes représentent les séquences, qui sont catégorisées selon leur quantité de changement, et les lignes reliant ces boîtiers représentent les sous-systèmes.

« … *Ce type de couplage ajoute plus de détail au couplage logique entre les sous-systèmes et identifie les programmes et les versions spécifiques dans lesquels les sous-systèmes exhibent exactement le même modèle de changement* ».

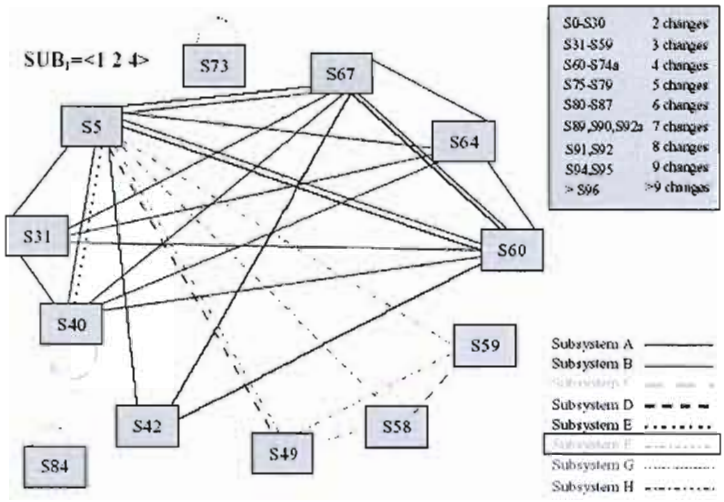


Figure A.36 Couplage entre les séquences

A.2.6.2 CRA

Chaque programme à plusieurs rapports de changements chacun propre à une version donnée du système. Le tableau ci-dessous présente les éléments caractérisant un rapport de changement original.

En ce qui concerne cette étude de cas, les rapports de changements se rapportent généralement à trois principaux types de changements :

1. *Développement ultérieur* [88](*Further Development FD*) qui est subdivisé en plusieurs parties telles que FD basé sur les spécifications système, FD basé sur le développement de technologie (logiciel et matériel) et ainsi de suite.

2. *Changement* général comme l'optimisation ou l'amélioration.

---

[88] Si deux programmes ont le même FD, cela veut dire qu'il y a une dépendance quelque part entre les deux programmes.

Correction d'une erreur qui se rapporte à un *rapport de bogue* [89](*Bug report BR*) spécifique

Tableau A.27 Rapport de changement générique

Rapport de changements
Endroit : identificateur du changement du document ou du module.
Date : quand le changement a été fait.
Symptôme : type du changement.
Résultat final : succès du changement comme montré par les tests.
Mécanisme : comment et par qui le changement a été exécuté.
Cause : corrective, adaptative, préventive ou perfective.
Sévérité : impact sur le reste du système, parfois indiqué sur une échelle de mesure ordinale.
Coût : temps et effort requis pour la mise en œuvre et le test du changement.

Ce type de rapport est utilisé pour s'assurer que les couplages logiques déjà identifiés entre les différents programmes, lors du processus CSA, ne soient pas dus à de simples coïncidences. Pour cela, on établit une analyse détaillée de ces rapports, suivant ces étapes :

- On inspecte les numéros de version de programmes ainsi que les séquences de changement.
- On liste les rapports de changement qui décrivent une nouvelle version d'un programme.

A.2.6.3 Références

Les auteurs ont utilisé vingt-quatre (24) références :

[89] Si deux programmes ou plus se réfèrent au même rapport de bogue, ils sont fortement dépendants l'un de l'autre, car la même erreur doit être corrigée dans les deux.

- Deux livres, un qui date des années 1980 et le livre de Yourdon et Constantine (1979).
- Vingt-deux articles, deux des années 1980 et vingt des années 1990.

A.2.7 Allen et Khoshgoftaar (1999) « Mesure de la cohésion et du couplage : Une approche de théorie de l'information »

Puisqu'on est dans la phase conception, les auteurs se basent surtout sur des graphes[90] — plutôt que du code comme dans bien d'autres études.

« *Plusieurs mesures de complexité du logiciel ont été proposées mais peu de mesures de cohésion et de couplage ont vu le jour jusqu'à présent*[91]. *Les récentes recherches se sont concentrées sur les divers types de cohésion et de couplage dans la conception orientée objet*[92] ».

La cohésion et le couplage sont deux attributs mesurables, qui utilisés avec des mesures d'autres attributs, prennent part dans l'évaluation et la prédiction de la qualité, puisqu'ils reflètent « *le degré de connectivité entre les sous-systèmes et dans le même sous-système respectivement*[93] ».

Briand, Morasca, et Basili (1996), ont mis en œuvre un cadre formel pour définir plus précisément le couplage et la cohésion d'un système modulaire, ce cadre est une sorte d'ensemble de propriétés.

Le tableau ci-dessous présente les propriétés du couplage inter-modules. Avec les changements suivants le tableau décrit aussi les propriétés du couplage intra-modules. :

1. On remplace couplage par couplage intra-module,

---

[90] Les graphes sont utilisés lors de la phase de conception comme une abstraction du logiciel, représentant ses différents composants et les relations existantes entre ces derniers.
[91] On classe bien évidemment les mesures de cohésion et de couplage dans la case de mesures de complexité du logiciel. Voir III.2.
[92] Chose que je ne peux ni appuyer ni contester pour le moment !
[93] Définitions de cohésion et couplage telles que données par les auteurs de l'article.

2. on remplace inter par intra, et

3. dans la propriété : fusionnement de modules, on remplace « plus grand que » par « plus petit que ».

Tableau A.28 Propriétés du couplage

Propriétés du Couplage d'un système modulaire
Non négatif : Le couplage d'un système modulaire est non négatif.
Valeur nulle : Le couplage d'un système modulaire est nul si son ensemble d'arêtes inter-modules est vide.
Monotonicité : Ajouter une arête inter-modules à un système modulaire ne diminue pas son couplage.
Fusionnement des modules : Si deux modules, m1 et m2, sont fusionnés pour former un nouveau module, m12, qui remplace m1 et m2, alors le couplage du système modulaire avec m12 n'est pas plus grand que le couplage du système modulaire avec m1 et m2 séparés.
L'additivité de modules disjoints : Si deux modules, m1 et m2, qui n'ont aucune arête inter-modules entre leurs sommets, sont fusionnés pour former un nouveau module, m12, qui remplace m1 et m2, alors Le couplage du système modulaire avec m12 est égal au couplage du système modulaire avec m1 et m2.

Le tableau ci-dessous présente les propriétés de la cohésion.

Tableau A.29 Propriétés de cohésion

Propriétés de cohésion d'un système modulaire
Non négativité et *Normalisation* : La cohésion d'un système modulaire appartient à un intervalle spécifié, Cohésion (MS) $\in$ [0, Max].
Valeur nulle : La cohésion d'un système modulaire est nulle si son ensemble d'arêtes internes aux modules est vide.
Monotonicité : Ajouter une arête interne d'un module à un système modulaire ne diminue pas sa cohésion.
Fusionnement des modules : Si deux modules indépendants, m1 et m2, sont fusionnés pour former un nouveau module, m12, qui remplace m1 et m2, alors la cohésion du système modulaire avec m12 n'est pas plus grande que la cohésion du système modulaire avec m1 et m2.

Avant d'introduire les mesures, les auteurs reprennent la définition de protocole de mesure de Kitchenham, Pfleeger et Fenton (1995) : « *un ensemble de conditions qui assurent un mesurage cohérent et pouvant être répété d'un attribut* » et celle d'instrument de mesure : « *un outil de mappage d'une abstraction de logiciel à un nombre ou à une catégorie* ». Ils ajoutent qu'un protocole de mesure est valide s'il est : « *1) Indépendant de celui qui mesure et de l'environnement de mesure et 2) Compatible avec l'unité de mesure désirée et le but de l'activité de mesurage* ».

Les auteurs ont choisi comme protocole de mesure pour leurs mesures n'importe quel protocole qui a comme conséquence une abstraction graphique d'un système modulaire, et puis traduisons cela en table sommets * arêtes inter(ou intra)-modulaires. Ils représentent les modules[94] du système sous forme de boîtes, chacune de ces boîtes comprend des sommets liés entre eux par des arêtes appelées des arêtes inter-modulaires ou intra-modulaires.

Le sommet déconnecté marqué par un 0 représente l'environnement du système, les auteurs tiennent à la représentation de l'environnement du système pour marquer leur désintérêt vis-à-vis des différentes interfaces du système.

Pour le couplage inter-modulaire, on ne fait que représenter le système avec toutes les liaisons qui existent entre ses modules, et on traduit ce graphe par une table sommets * arêtes inter-modulaires comme on le voit dans la figure ci-dessous[95].

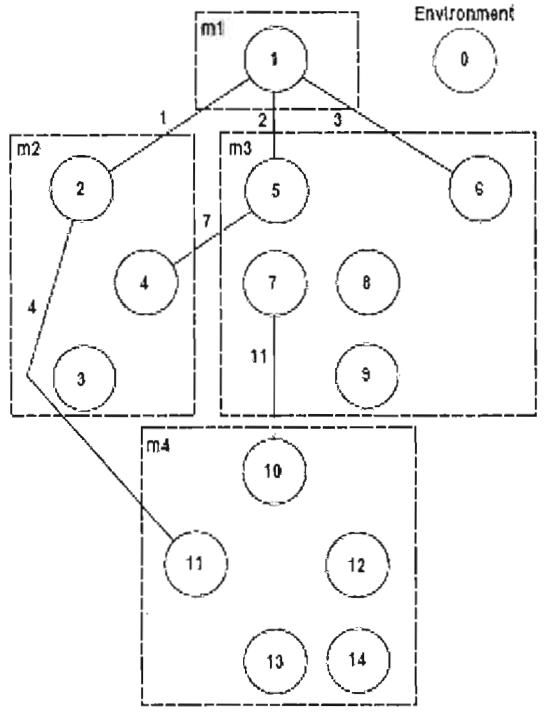
Sur les lignes on présente les numéros des sommets et sur les colonnes on présente les numéros des arêtes reliant ces sommets, si un sommet donné est lié à une arête on met un 1 dans l'intersection de la ligne, de ce sommet et de la colonne de cette arête, sinon on met 0.

La dernière colonne PL(i), est calculée en divisant par 15 (nombre de lignes) le nombre d'occurrences du même patron. Par exemple, le patron « tous des 0 » est présent 7 fois donc

---

[94] Pour les auteurs un module est un sous-système.
[95] Les exemples de graphes et de tables de relations sont tirés de l'article, ils ont les propriétés déjà mentionnées.

la valeur pour les sommets ayant 0 arêtes est 7/15 = 4,66666666. Il y a un seul patron avec « trois 1 » au début (sommet de m1) et donc 1/15= 0,067, et c'est la même valeur des autres patrons avec une seule occurrence.



Sous-Graphe d'arêtes inter-modulaires

Module	Node	Edge 1	2	3	4	7	11	$p_{L(i)}$
Environ.	0	0	0	0	0	0	0	0.467
m_1	1	1	1	1	0	0	0	0.067
m_2	2	1	0	0	1	0	0	0.067
	3	0	0	0	0	0	0	0.467
	4	0	0	0	0	1	0	0.067
m_3	5	0	1	0	0	1	0	0.067
	6	0	0	1	0	0	0	0.067
	7	0	0	0	0	0	1	0.133
	8	0	0	0	0	0	0	0.467
	9	0	0	0	0	0	0	0.467
m_4	10	0	0	0	0	0	1	0.133
	11	0	0	0	1	0	0	0.067
	12	0	0	0	0	0	0	0.467
	13	0	0	0	0	0	0	0.467
	14	0	0	0	0	0	0	0.467

Table de nœuds * arêtes inter-modulaires

Figure A.37 Exemple de représentation graphique et tabulaire inter-modulaire d'un système modulaire

Quant au couplage intra-modulaire et la cohésion, on se base sur un graphe représentant tous les modules du système et qui expose les liaisons existantes entre les éléments de chaque module, de même ce graphe est traduit par une table de sommets * arêtes intra-modulaires, comme on le voit dans la figure ci-dessous.

D'autre part, les auteurs ont adopté la théorie de l'information comme base pour leur instrument de mesure des trois mesures présentées. Car ils voient les décisions conceptuelles contenues dans les sommets comme de l'information.

La capacité de la mémoire à court terme étant limitée[96], quand un concepteur prend une décision, il ne considère pas nécessairement toute l'information dont il aurait besoin ce qui peut créer des erreurs de conception.

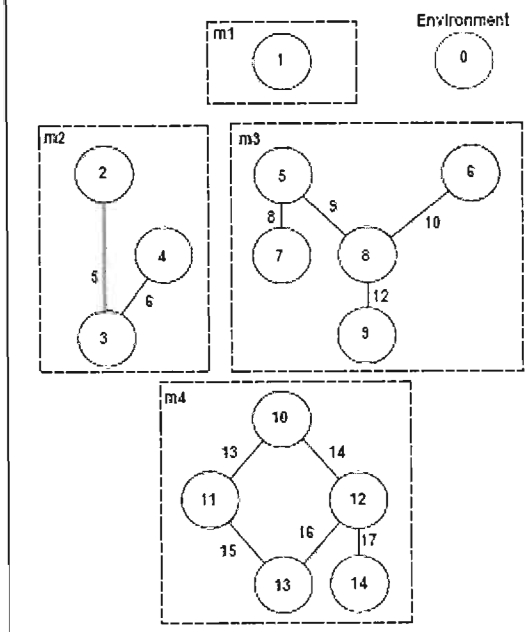
« *Notre hypothèse de fonctionnement est l'évaluation des conceptions en termes de quantités de divers types d'information pouvant indiquer la surcharge potentielle de l'information* ».

Les auteurs proposent une approche sophistiquée qui dépasse le comptage d'arêtes comme font d'anciennes métriques.

« *La théorie de l'information est attrayante, parce qu'elle mesure le contenu symbolique loin de compter la redondance et les patrons. Selon la théorie de l'information, les patrons réguliers ont un contenu d'information faible, parce que le contenu symbolique peut être décrit de façon compacte* ».

---

[96] Le célèbre 7, nombre magique de Miller (Voir III.2).



Sous-graphe d'arêtes intra-modulaire

Module	Node	Edge 5	6	8	9	10	12	13	14	15	16	17	$p_{L(i)}$
Environ.	0	0	0	0	0	0	0	0	0	0	0	0	0.133
m_1	1	0	0	0	0	0	0	0	0	0	0	0	0.133
m_2	2	1	0	0	0	0	0	0	0	0	0	0	0.067
	3	1	1	0	0	0	0	0	0	0	0	0	0.067
	4	0	1	0	0	0	0	0	0	0	0	0	0.067
m_3	5	0	0	1	1	0	0	0	0	0	0	0	0.067
	6	0	0	0	0	1	0	0	0	0	0	0	0.067
	7	0	0	1	0	0	0	0	0	0	0	0	0.067
	8	0	0	0	1	1	1	0	0	0	0	0	0.067
	9	0	0	0	0	0	1	0	0	0	0	0	0.067
m_4	10	0	0	0	0	0	0	1	1	0	0	0	0.067
	11	0	0	0	0	0	0	1	0	1	0	0	0.067
	12	0	0	0	0	0	0	0	1	0	1	1	0.067
	13	0	0	0	0	0	0	0	0	1	1	0	0.067
	14	0	0	0	0	0	0	0	0	0	0	1	0.067

Table de nœuds * arêtes intra-modulaires

Figure A.38 Exemple de représentation graphique et tabulaire intra-modulaire d'un système modulaire

A.2.7.1 Couplage inter-module

Couplage inter-module : Soit un système modulaire donné, MS, et son sous-graphe d'arêtes inter-modules, S, le couplage inter-module de ce système est la longueur minimale de description des relations dans S.

$$\text{Couplage (MS)} = \sum I(Si) - I(S) \text{ (i allant de 1 à n)}$$

Avec :

- I(Si) : la longueur de description minimale estimée de Si
- I(S) : la longueur de description minimale estimée de S

Maintenant on définit I(S) et I(Si), Si est le sous-graphe composé de tous les sommets et arêtes de S et qui ont le $i^{ème}$ sommet comme point final.

$I(S) = (n+1)\, H(S)$

$I(Si) = (n+1)\, H(Si)$

Avec :

- n: le graphe du système modulaire est constitué de n sommets
- H(S) : L'entropie[97] de la distribution des étiquettes de sommet de S
- H(Si) : L'entropie de la distribution des étiquettes de sommet de Si

H(S) est :

$$H(S) = \sum(-pl \log pl) \quad (l \text{ allant de } 1 \text{ à } ns)$$

$$H(S) = \sum [1/(n+1)] * [-\log PL(i)] \quad (i \text{ allant d } 0 \text{ à } n)$$

De même H(Si) est :

$$H(Si) = \sum [1/(n+1)] * [-\log PLi(j)] \quad (j \text{ allant de } 0 \text{ à } n)$$

Avec :

- ns : le nombre d'étiquettes distinctes
- L(i) : la fonction qui donne l'index de modèle l de la $i^{ème}$ ligne de S

---

[97] L'entropie de Shannon, est une fonction mathématique qui correspond à la quantité d'informations contenue ou délivrée par une source d'information. Cette source peut être une langue, un signal électrique, ou un fichier informatique quelconque. La définition de l'entropie d'une source selon Shannon est telle que plus la source est redondante, moins elle contient d'information. En l'absence de contraintes particulières, l'entropie est ainsi maximale pour une source dont tous les symboles son équiprobables.

- Li(j) : la fonction qui donne l'index de modèle l de la $j^{ème}$ ligne de Si

D'où, I(S) et I(Si) deviennent :

$$I(S) = \sum (-\log PL(i)) \quad (i \text{ allant de 1 à n})$$

$$I(Si) = \sum (-\log PLi(j)) \quad (j \text{ allant de 1 à n})$$

A.2.7.2 Couplage intra-module

Le couplage intra-module est défini comme suit : Soit un système modulaire donné, MS, et son sous-graphe d'arêtes intra-modulaires, S`, le couplage intra-module de ce système est la longueur minimale de description des relations dans S`.

$$\text{Couplage Intramodule (MS)} = \sum I(S`i) - S`(I) \quad (i \text{ allant de 1 à n})$$

A.2.7.3 Cohésion

Soit un système modulaire, MS, et son sous-graphe d'arêtes intra-modulaires, S`. On considère le système modulaire MS(n), se composant d'un graphe complet de n sommets contenus dans un seul module, où chaque sommet est relié à tous les autres nœuds. MS(n) est le système le plus cohésif possible avec n sommets, et son sous-graphe S(n) représente tous les sommets et arêtes du système. La cohésion du système MS est la longueur minimale de description des relations dans S` divisée par la longueur minimale de description des relations dans S(n).

$$\text{Cohésion (MS)} = \text{Couplage Intramodule(MS)/Couplage Intermodule (MS(n))}$$

Il a été prouvé que ces trois mesures sont des mesures théoriquement valides, puisqu'elles ont les propriétés d'une mesure de logiciel théoriquement valide proposées par Kitchenham, Pfleeger et Fenton (1995).

A.2.7.4 Références

Les auteurs ont utilisé trente-sept (37) références :

- Deux livres qui datent des années 1990, dont un est celui de Pressman (1992).
- Trente-quatre articles:
    - Un qui date des années 1940
    - Un qui date des années 1950
    - Un des années 1960
    - Trois des années 1970
    - Un des années 1980, il s'agit de l'article de Weyuker (1988) que nous avons résumé dans l'Appendice C.
    - Vingt-sept publiés en 1990, un d'entre eux a été synthétisé, c'est celui de Bieman et Kang (1998).
- Une thèse qui date des années 1990.

A.3 Couplage et cohésion : Années 2000-2008

Dans cette section, nous résumons sept (7) articles traitant le couple C&C publiés dans les années 2008-2008. Ces articles sont :

1. « Mesure de la cohésion pour les classes orienté-objet »
2. « Mesure du couplage et de la cohésion des modules logiciels : Une approche basée sur la théorie de l'information »
3. « Complexité de logiciel : Vers une théorie unifiée du couplage et de cohésion »
4. « Métriques de cohésion basées sur le tranchage et intervention sur le logiciel »
5. « Détection du couplage indirect »
6. « Pour visualiser le couplage entre les modules »
7. « L'évaluation des principes de couplage et cohésion de paquet pour la prédiction de la qualité de conception orientée objet »

A.3.1 Chae, Kwon et Bae (2000) « Mesure de la cohésion pour les classes orienté-objet »

A.3.1.1 Introduction

Avec la diffusion du paradigme de programmation OO et son adoption comme méthodologie d'analyse et de conception dans l'industrie et dans les milieux académiques, plusieurs chercheurs ont conçu des mesures de cohésion OO. Dans leur article, Chae, Kwon et Bae (2000) présentent les plus importantes métriques de cohésion OO existantes avec leurs définitions. Ces métriques sont définies dans le tableau ci-dessous

Tableau A.30 Les métriques de cohésion OO existantes

Métrique	Définition
LCOM1 (*Lack of Cohesion in Methods*)	Nombre de paires de méthodes qui ne partagent aucune variable d'instance.
LCOM2	Soit P, les paires de méthodes avec aucune variable d'instance partagée, et Q, les paires de méthodes avec des variables d'instance partagées. LCOM2 = $\|P\| - \|Q\|$, si $\|P\| > \|Q\|$; 0 autrement
LCOM3	On considère un graphe G non orienté dont les sommets sont les méthodes de classe. Il existe une arête entre deux sommets si les méthodes correspondantes partagent au moins une variable d'instance. LCOM3 = |Composants connectés de G|
LCOM4	Similaire à LCOM3, sauf que le graphe G a en plus une arête entre les sommets représentants des méthodes Mi et Mj, si Mi appelle Mj ou *vice-versa*.
Co	Soit V les sommets du graphe G de LCOM4, et E ses arêtes. Alors : Co = 2*[(|E| - (|v| - 1)) / ((|v| -1)*(|v| -2))]
LCOM5	On considère un ensemble de méthodes {Mi} (i = 1, …, m) accédant à un ensemble de variables d'instance {Aj} (j = 1, …, a). Soit ϕ(Aj) le nombre de méthodes qui référencient Aj. Alors : LCOM5= $((1/a) \sum \phi(Aj) - m)/(1-m)$ $1 \leq j \leq a$
Coh	Une variation sur LCOM5. Coh = $\sum \phi(Aj)/(m*a)$ $1 \leq j \leq a$
LCC	Soit NP le nombre maximum de paires de méthodes publiques.

Métrique	Définition
	NP =[N * (N -1)]/2, pour N méthodes publiques. Soit NIC le nombre de connexions directes et indirectes entre les méthodes publiques. Alors LCC est définie comme le nombre relatif de méthodes publiques connectées directement et indirectement. LCC = NIC/NP
TCC	Soit NDC le nombre de connexions directes entre les méthodes publiques. Alors TCC est définie comme le nombre relatif de méthodes publiques connectées directement. TCC = NDC / NP

A.3.1.1 Méthodes spéciales

Pour mieux cerner la cohésion les auteurs introduisent les concepts de « méthodes spéciales » qu'ils organisent en 3 (trois) types :

1. *Méthode d'accès* : un type de méthode dont le comportement se résume à la lecture ou à la mise à jour d'une variable d'instance (*get* et *set*).

2. *Méthode de délégation* : un type de méthode qui réalise sa tâche dans une classe en transmettant un message à un autre objet.

3. *Constructeur&Destructeur* : un constructeur et un destructeur d'une classe doivent accéder aux variables d'instance essentielles qui requièrent une initialisation et une libération.

Dans le tableau ci-dessous, on présente les problèmes que peut causer ce genre de méthodes spéciales et leurs influences sur les métriques de cohésion exposées auparavant.

Tableau A.31 Analyse des métriques de cohésion existantes

Méthode spéciale	Impact sur la cohésion	Métriques de cohésion affectées
Méthode d'accès	Généralement, ces méthodes ne réduisent pas la cohésion parce qu'elles ne font que référencer une variable d'instance dans une classe donnée. Seulement, les métriques existantes impliquent ces méthodes dans le calcul de la cohésion, ce qui mène à une réduction inattendue de la cohésion. Nous mentionnons que certains langages de programmation génèrent automatiquement ces méthodes.	Augmentation de LCOM1 et LCOM2. Diminution de Co et TCC, car ces méthodes augmentent le nombre de paires de méthodes qui ne partagent aucune variable d'instance. Augmentation de LCOM5 et diminution de Coh, car le nombre d'interactions possibles augmente avec le nombre de variables d'instance.
Méthode de délégation	Ces méthodes interagissent avec certaines variables d'instance. Par conséquent, la cohésion d'une classe ne devrait pas être réduite par l'existence de ces méthodes. Mais les mesures de cohésion courantes n'en tiennent pas compte.	Non spécifié.
Constructeur/Destructeur	Bieman et Kang (1995) n'ont pas considéré les constructeurs et les destructeurs comme membres d'une classe lors de l'application des métriques LCC et TCC sur cette dernière, et cela dans le but d'enlever l'impact de connexion artificielle de ces méthodes.	Non spécifié.

A.3.1.2 Critères de cohésion

D'après les auteurs, les métriques de cohésion existantes dépendent de l'utilisation ou bien du partage des variables d'instance. En effet, une classe a une cohésion forte si :

- Un grand nombre de ses variables d'instance est référencé par une méthode (par exemple, LCOM5 et Coh), ou
- Un grand nombre de paires de méthodes partagent des variables d'instance (par exemple LCOM1, LCOM2, LCOM3, LCOM4, Co, LCC, et TCC).

Les métriques qui appliquent ces critères peuvent donner des résultats qui ne sont pas utiles ou carrément incorrects. Pour cela les auteurs adoptent comme nouveau critère, « *la connectivité entre les membres d'une classe* ». D'après ce critère, une « *classe est hautement cohésive si ses membres sont fortement liés* ».

A.3.1.3 Nouvelle mesure de cohésion

A.3.1.3.1 Définitions de base

Les auteurs introduisent quelques définitions qui sont utilisées dans la définition de leur nouvelle mesure de cohésion :

Définition 1 : Une classe C comprend un ensemble de variables d'instance V(C) et un ensemble de méthodes M(C).

Définition 2 : Pour une méthode Mi, soit R(Mi) l'ensemble de variables d'instance qui sont directement référencées par Mi, et I(Mi) l'ensemble de méthodes qui sont directement appelées par Mi.

I*(Mi) est l'ensemble de méthodes directement ou indirectement appelées par Mi.

$$I^*(M_i) = \{m \in M(C) \mid m \in I(M_i) \vee \exists m_i \in I(M_i) \cdot m \in I^*(m_i)\}$$

Et, R*(Mi) est l'ensemble des variables d'instance directement ou indirectement référencées par Mi.

$$R^*(M_i) = \{v \in V(C) \mid v \in R(M_i) \vee \exists m_i \in I^*(M_i) \cdot v \in R(m_i)\}$$

Définition 3 : Une méthode dans une classe est spéciale si c'est une méthode d'accès, une méthode de délégation, un constructeur, ou un destructeur. Une méthode est dite normale si elle n'est pas spéciale. Ainsi, M(C) peut être divisé en deux ensembles : ensemble de méthodes spéciales Ms(C) et ensemble de méthodes normales Mn(C).

$$M_s(C) \cap M_n(C) = \emptyset \text{ and } M_s(C) \cup M_n(C) = M(C).$$

Définition 4 : Un graphe de référence d'une classe C est une représentation des interactions entre les membres de cette classe, et est défini comme un graphe non orienté G (N, A) avec :

- $N = V(C) \cup M(C)$;
- $A = \{(m, v) \mid v \in R^*(m) \text{ where } m \in M(C) \text{ and } v \in V(C)\}$.

Définition 5 : Un graphe de référence Gr (N, A) est le composant le plus cohésif (*Most Cohesive Component* MCC), si chaque méthode normale[98] dans Mn(Gr) a des interactions avec toutes les variables d'instance dans V(Gr).

$$A = \{(m, v) \mid m \in M_n(G_r), v \in V(G_r)\}$$

Avec :

Mn(Gr) : l'ensemble de méthodes normales dans Gr.

V(Gr) : l'ensemble de variables d'instance dans Gr.

A.3.1.1.1 Connectivité des membres d'une classe

Tel que déjà énoncé, les auteurs croient que la cohésion d'une classe ne dépend pas seulement du nombre d'interactions internes, mais aussi des caractéristiques de ces interactions.

Une classe a un sous-ensemble de méthodes sans lesquelles son graphe de référence Gr devient disjoint. On appelle ce sous-ensemble « méthodes adhésives » (*glue methods*), notées

---

[98] On ne parle pas des méthodes spéciales car ces dernières n'ont pas à interagir avec toutes les variables d'instance d'une classe.

Mg(Gr). Pour montrer comment les membres d'une classe sont fortement connectés par les méthodes adhésives, on introduit la notion de « facteur de connectivité ».

Définition 6 : Le facteur de connectivité d'un graphe de référence Gr, Fc(Gr), représente le degré de connectivité parmi les membres. C'est le rapport entre le nombre de méthodes adhésives et le nombre de méthodes normales[99].

$$F_c(G_r) = \frac{|M_g(G_r)|}{|M_n(G_r)|}$$

Avec :

Mg(Gr) : le nombre de méthodes adhésives dans Gr.

Mn(Gr) : le nombre de méthodes normales dans Gr.

Les valeurs de Fc, sont entre **0** et 1.

A.3.1.1.2 Structure hiérarchique des interactions des modèles

Les auteurs croient que la cohésion d'une classe est affectée par les cohésions entre ses composants dans l'arbre de structure (*structure tree*).

Définition 7 : L'arbre de structure d'un graphe de référence Gr, Ts(Gr), est un arbre T= (N,A), avec la racine Gr, et où :

- $N = \{G_r^i \mid G_r^i \subseteq G_r\}$;
- $A = \{(G_r^P, G_r^c) \mid G_r^c$ is one of the connected sub-reference graphs obtained from G_r^P by removing $M_g(G_r^P)\}$;

[99]Les méthodes spéciales n'ont aucune influence sur le facteur de connectivité.

De ce fait, les auteurs introduisent la notion de facteur de structure qu'ils définissent comme suit :

Définition 8 : Le facteur de structure pour un graphe de référence Gr, Fs(Gr), est défini comme la cohésion moyenne de ses fils dans l'arbre.

$$F_s(G_r) = \frac{1}{n}\sum_{i=1}^{n} CBMC(G_r^i)$$

Où, G i r est un des n enfants de Gr dans l'arbre, et

CBMC (G i r) dénote la cohésion d'un composant G i r.

En utilisant le facteur de connectivité et le facteur de structure, on définit la nouvelle métrique CBMC :

Définition 9 : La CBMC d'une classe C, CBMC(C), est définie comme le facteur de connectivité de son graphe de référence Fc(Gr(C)), multiplié par le facteur de structure de son graphe de référence Fs(Gr(C)).

$$\begin{aligned} CBMC(C) &= F_c(G_r(C)) \times F_s(G_r(C)) \\ &= F_c(G_r(C)) \times \frac{1}{n}\sum_{i=1}^{n} CBMC(G_r^i(C)) \end{aligned}$$

« *Le facteur de connectivité d'une classe indique la force du rapport entre ses membres, et le facteur de structure dénote la contribution des composants à la cohésion globale de la classe* ». La nouvelle métrique de cohésion CBMC possède les propriétés suivantes :

- Les méthodes spéciales n'ont aucune influence sur la cohésion.
- La cohésion d'une classe dépend du degré de connectivité de ses membres.

- Une classe avec des composants faiblement cohésifs est de faible cohésion.

A.3.1.1.3 Application de la mesure de cohésion

Les facteurs de connectivité et de structure employés dans le calcul de la métrique de cohésion CBMC « *peuvent capturer un certain aspect de la conception des classes* ». En combinant ces deux facteurs, la qualité de conception peut être mieux étudiée. Les auteurs identifient quatre (4) catégories de classes selon que leurs facteurs de connectivité et de structure soient bas ou élevés. Ces catégories sont :

- **Classe à cohésion forte (Fc élevé et Fs élevé) :** une classe de ce type est considérée comme une classe idéale, car ses membres sont étroitement liés par un certain nombre de méthodes adhésives. Et même quand elle est divisée en plusieurs composants, chacun des composants a une cohésion élevée.

- **Classe à restructurer (Fc faible et Fs élevé) :** Une classe de ce type peut être facilement subdivisée en plusieurs composants hautement cohésifs et cela en enlevant quelques méthodes adhésives. La classe originale peut être restructurée en créant une nouvelle classe pour chaque composant, ayant comme variables d'instance, les variables d'instance de chaque composant.

- **Classe à cohésion faible (Fc élevé et Fs faible) :** les membres d'une classe de ce type sont fortement liés par un certain nombre de méthodes adhésives. Cependant les composants divisés peuvent être facilement dédoublés en retirant quelques unes de ces méthodes. Cette catégorie de classes indique une faible conception, mais ce n'est pas un cas courant en pratique.

- **Classe de conception hasardeuse (Fc bas et Fs bas) :** une classe de cette catégorie n'est autre qu'une collection de membres qui ont peu de relations entre eux. L'existence de ce genre de classe est inutile, et ses membres doivent être replacés dans les classes appropriées.

A.3.1.2 HYSS : outil de mesurage de la cohésion :

Les auteurs ont développé un outil de mesure de la cohésion appelé HYSS. Cet outil automatise le calcul des métriques de cohésion prises en considération par les auteurs, ainsi que la nouvelle métrique CBMC qu'ils proposent. Il est destiné aux programmes écrits en C++.

HYSS accepte en entrée un programme source C++ duquel il extrait les informations concernant ses classes, comme : les variables d'instance, les méthodes et les interactions entre ces dernières, en utilisant GEN++. À partir de ces informations, un graphe de référence propre à chaque classe est créé. Par la suite les méthodes adhésives sont identifiées en essayant de décomposer le graphe de référence. Le facteur de connectivité correspond au nombre de méthodes adhésives, et le facteur de structure est calculé à partir de la moyenne de CBMC des composants constituant le graphe de référence, et cela en appliquant l'algorithme à chaque composant.

La figure ci-dessous présente l'algorithme qui décrit le calcul du CBMC d'un graphe de référence :

```
function CBMC(G_r)
  switch G_r of
     case G_r is MCC : return 1
     case G_r is disjoint : return 0
     otherwise : // G_r is connected, but not an MCC
        for k = 1 to |M(G_r)| - 1
           for m = 1 to (|M(G_r)| choose k)
              select a different set of k methods from G_r ⇒ M_g(G_r)
              if G_r can be decomposed into G_r^1, G_r^2, ..., G_r^n by removing M_g(G_r) then
                 F_{s_m} = (1/n) Σ_{1≤i≤n} CBMC(G_r^i)
              else
                 F_{s_m} = 0
              endif
           endfor
           if G_r was decomposed by removing k methods then
              F_c = k / |M_n(G_r)|
              F_s = Max_{1≤m≤(|M(G_r)| choose k)} F_{s_m}  // choose the maximum structure factor
              return F_c × F_s
           endif
        endfor
  endswitch
end
```

Figure A.39 Algorithme de calcul du CBMC d'un graphe de référence

A.3.1.2.1 Identification de méthodes spéciales

La nouvelle métrique proposée procède à un traitement unique des méthodes spéciales. Donc ces dernières doivent être correctement identifiées. Les constructeurs et les destructeurs sont facilement détectés, il suffit simplement de voir la syntaxe des classes C++. En ce qui concerne les méthodes d'accès, elles ont habituellement une des deux formes suivantes :

- Instruction d'affectation : variable-d-instance=valeur
- Instruction de retour : return variable-d-instance

Les méthodes de délégation ont généralement la forme suivante :

Appel d'une méthode pour récupérer une variable d'instance : variable-d-instance [.|->] une-méthode ().

HYSS considère une méthode comme une méthode d'accès si elle comprend une seule instruction de retour, ou une instruction d'affectation référençant une seule variable d'instance, et la considère comme une méthode de délégation, si elle fait seulement un appel d'une méthode pour une variable d'instance.

La figure ci-dessous décrit la procédure pour reconnaître si une méthode donnée Mi dans une classe C est spéciale ou normale.

```
function KINDOFMETHOD(Method Mi, Class C)
  if ( name of Mi = name of C ) then return Constructor
  if ( name of Mi = ~ name of C ) then return Destructor

  RefCount = the number of instance variables referenced by method Mi
  StmtCount = the number of statements in method Mi
  StmtType = the kind of the single statement in method Mi
  if ( StmtCount = 1 ) and ( RefCount = 1 ) then
     if ( StmtType = Assignment Statement or Return Statement ) then
        return Access Method
     if ( StmtType = Method invocation on an instance variable ) then
        return Delegation Method
  endif
  return Normal Method
end
```

Figure A.40 Détermination des méthodes spéciales

A.3.1.2.2 Considérations des classes C++

Les adaptations suivantes ont été effectuées par les auteurs pour appliquer la nouvelle métrique de cohésion aux classes C++ :

- *Méthodes non-publiques* : seules les méthodes publiques sont considérées par la nouvelle métrique, car ce sont elles qui capturent le comportement et l'état des objets, tandis que les méthodes non-publiques sont indirectement considérées, puisque généralement elles sont introduites pour rendre la mise en œuvre des autres méthodes publiques plus facile.

- *Membres hérités* : les membres hérités sont considérés par la nouvelle métrique comme suit :

    - Toutes les variables d'instance héritées sont considérées.

    - Pour les méthodes héritées, seules celles qui peuvent être évoquées publiquement sont considérées.

A.3.1.3 Étude empirique :

Afin de démontrer l'efficacité de la nouvelle métrique CBMC, les auteurs procèdent à une étude empirique. Ils ont choisi comme système un système nommé *InterViews*, qui est en fait une trousse à outils de C++ pour *Windows*, développé à l'université de *Stanford*. Ce système fournit un ensemble de classes qui définissent le comportement des objets

graphiques d'interface utilisateur, tels que des fenêtres, des boutons, des menus et des documents.

La figure ci-dessous présente la distribution des variables d'instance dans les 134 classes du système *InterViews* utilisées dans l'étude. On remarque que la plupart (72 %) des méthodes comprennent un petit nombre de variables d'instance qui ne dépasse pas cinq (5) variables.

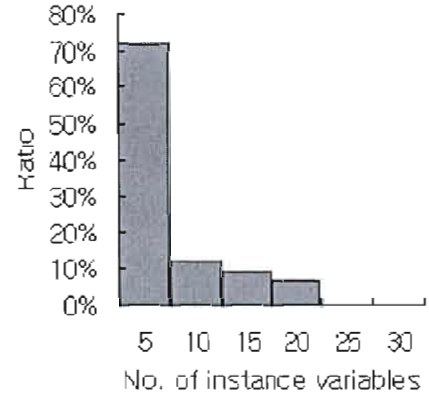


Figure A.41 Variables d'instance dans *InterViews*

La figure ci-dessous présente la distribution des méthodes dans *InterViews*. On trouve qu'environ 48 % des classes n'ont pas plus de dix méthodes, et quelques classes ont plus de cinquante (50) méthodes. On remarque aussi que l'exclusion des méthodes spéciales mène à l'augmentation du nombre de classes avec un nombre inférieur de méthodes. À peu près 52 % des méthodes n'ont pas plus de dix (10) méthodes, et aucune classe n'a plus de cinquante (50) méthodes.

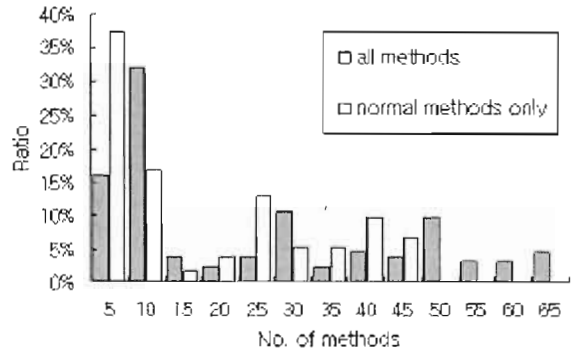


Figure A.42 Méthodes dans *InterViews*

Dans un premier lieu, les auteurs appliquent sur l'ensemble des classes l'algorithme de la figure 9.2 pour déterminer ses méthodes spéciales et ses méthodes normales. La figure ci-dessous montre les résultats de l'application de l'algorithme. On trouve qu'à peu prés 43 % des méthodes sont des méthodes spéciales. Ceci montre l'utilisation fréquente de méthodes spéciales dans des systèmes pratiques et appuie aussi la stratégie des auteurs dans le traitement de méthodes spéciales dans le calcul de la cohésion des classes.

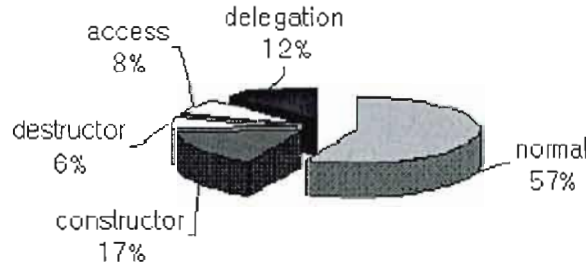


Figure A.43 Types de méthodes dans *InterViews*

Après l'identification des méthodes spéciales, les auteurs passent à celle des méthodes adhésives. La figure ci-dessous montre la distribution de ce type de méthodes dans

InterViews. On remarque que 70% des classes ont moins de quatre (4) méthodes adhésives et le nombre maximum de méthodes adhésives dans une classe est six (6).

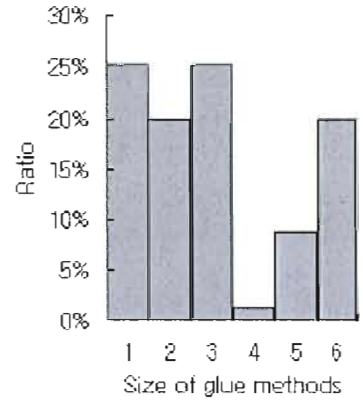


Figure A.44 Méthodes adhésives dans *InterViews*

A.3.1.3.1 Résultats du mesurage

La figure ci-dessous montre la distribution du facteur de connectivité, du facteur de structure et de la CBMC des classes du système *InterViews*. À partir de ces résultats, on remarque que « *plusieurs classes de ce système ont une cohésion très faible et cela est dû à leurs facteurs de connectivité très faibles* ».

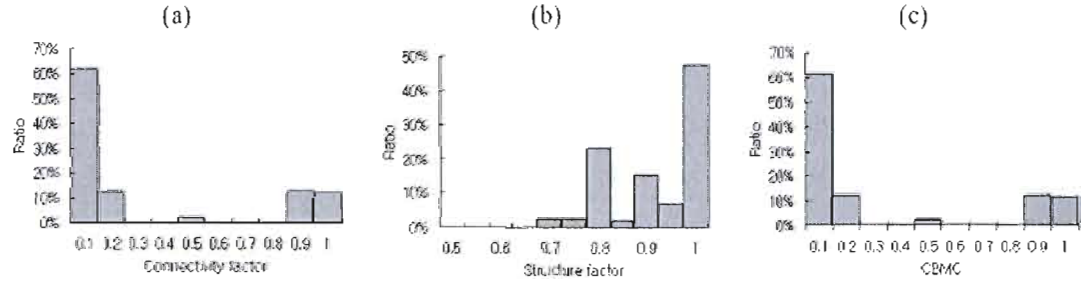


Figure A.45 (a) Facteur de connectivité (b) Facteur de structure (c) CBMC

La figure ci-dessous présente la classification des classes selon leurs facteurs de connectivité et leur facteur de structure comme a été décrit auparavant dans la section Application de la mesure de cohésion. Il n'y a aucune classe du troisième ou quatrième cas, en revanche il y en a beaucoup du premier et surtout du deuxième cas. En effet, presque 74 % des classes étudiées révèlent le premier cas. La ci-dessous montre la distribution de ces classes dans quatre différents types, qu'on prend le soin de détailler, comme suit :

- **Pas de méthodes normales** : Le système *InterViews*, contient huit classes n'ayant aucune méthode et cinq n'ayant aucune méthode normale, faisant en tout treize, ce qui constitue 10 % des classes de ce système.

- **Pas de variable d'instance** : « *une classe sans variables d'instance semble être une collection de quelques méthodes indépendantes et non pas une représentation d'objets* ». Comme dans le cas de la classe *osMath* d'*InterViews*, qui est une collection de fonctions arithmétiques et qui comprend vingt-cinq (25) méthodes.

- **Membres isolés** : plusieurs classes d'*InterViews* contiennent une ou plusieurs méthodes isolées, les auteurs jugent qu'une méthode est isolée « *si elle n'a aucune interaction avec les variables d'instance dans la classe* [...] *souvent elle reflète le comportement seulement avec les paramètres* ». Pourtant, « *il n'est pas raisonnable d'encapsuler dans une classe des méthodes isolées qui montrent un comportement indépendant des membres restants de la classe. Ainsi, l'existence de méthodes isolées est indésirable* ». De même, il y a des classes qui contiennent des variables d'instance isolées. Une variable d'instance isolée est définie comme « une variable qui n'est mise en référence par aucune méthode de la classe ».

- **Pas d'interaction avec les membres hérités** : les auteurs ont trouvé plusieurs classes qui n'ont aucune interaction avec les membres hérités des super-classes. On se retrouve face à ce genre de problème quand la super-classe et les sous-classes représentent des aspects différents.

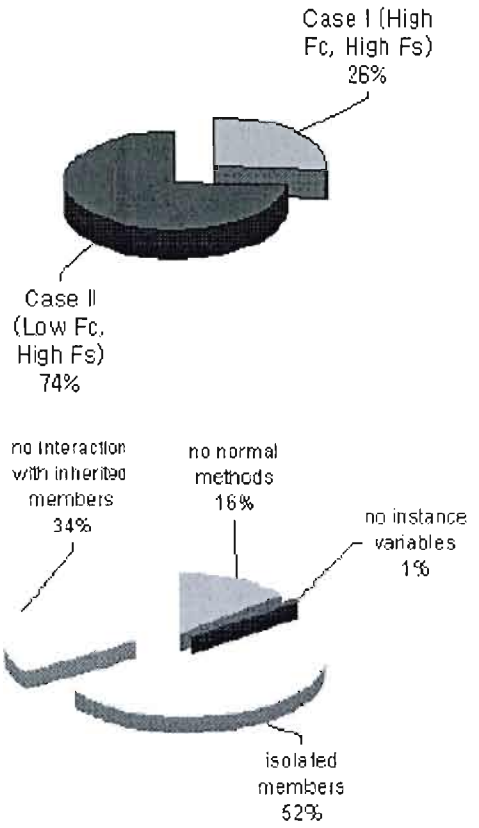


Figure A.46 CBMC des classes d'*InterViews* (a) catégories des classes (b) examen détaillé du 2ème cas

A.3.1.3.2 Analyse du composant principal

Pour démontrer que la métrique CBMC proposée est différente des métriques de cohésion existantes, les auteurs font une analyse statistique de type PCA (*Principal Component Analysis).* PCA est « *une technique standard d'identification des relations entres les variables dans un ensemble de données* ».

A.3.1.3.2.1 Procédures d'analyse

Collecte de données pour les métriques de cohésion :

En utilisant HYSS, on calcule les valeurs des métriques déjà décrites (LCOM1-5, Co, Coh, LCC, and TCC) pour toutes les classes du système *InterViews*.

Identification des « outliers » :

L'inclusion ou exclusion des *outliers* « *peut avoir une grande influence sur les résultats d'analyse* ». Donc, on les repère puis on les enlève.

Exécution du PCA:

Voir l'article pour les détails.

A.3.1.3.2.2 Les résultats des mesures :

Les auteurs appliquent les métriques sur 134 classes d'*InterViews*. Parmi ces 134 classes ils sont arrivés à déceler sept (7) *outliers*, donc, il leur reste 127 classes. Les statistiques descriptives des résultats sont présentées dans le tableau ci-dessous.

Tableau A.32 Statistiques descriptives pour des métriques de cohésions[100]

Measure	Max.	75%	Median	25%	Min.	Mean	Std. Dev.
LCOM1	819	12	2	0	0	30.93	113.54
LCOM2	777	0	0	0	0	19.73	92.71
LCOM3	30	2	1	1	1	2.83	4.41
LCOM4	25	2	1	1	1	1.88	2.56
Co	1	1	1	0.91	0.33	0.90	0.16
LCOM5	0.94	0.55	0.42	0.32	0	0.41	0.24
Coh	1	0.67	0.62	0.02	0	0.48	0.34
LCC	1	1	1	1	0	0.88	0.29
TCC	1	1	1	0.7	0	0.83	0.30
CBMC	1	0.5	0	0	0	0.25	0.39

Les auteurs ont identifié cinq PC qui décrivent 94.6% de la variance dans l'ensemble de données. Le tableau ci-dessous présente ces PC, qu'on décrit comme suit :

[100] Les principaux composants pour chaque PC sont encadrés.

PC1 (48.2%) : les métriques LCC, TCC et Co, dépendent généralement du nombre de paires de méthodes qui traitent directement ou indirectement les mêmes variables d'instance.

PC2 (27.5%) : LCOM1, LCOM2 et LCOM3, calculent le nombre de paires de méthodes qui ont des variables d'instance communes.

PC3 (7.9%) : LCOM5 et Coh, sont concernées par le nombre d'interaction entre les variables d'instance et les méthodes.

PC4 (5.9%) : CBMC dépend du degré de connectivité entre les membres d'une classe en excluant les méthodes spéciales.

PC5 (5.1%) : LCOM4, calcule le partage des variables d'instance en considérant celui fait indirectement par l'appel de méthodes.

À partir de cette analyse, les auteurs remarquent que la métrique CBMC est « *le seul facteur majeur dans PC4, cela veut dire que cette métrique capture un aspect différent des propriétés des classes* ».

Tableau A.33 Composants tournés

	PC_1	PC_2	PC_3	PC_4	PC_5
Eigenvalue	4.82	2.75	0.80	0.59	0.52
Proportion	48.2%	27.5%	7.9%	5.9%	5.1%
Cumulative	48.2%	75.7%	83.6%	89.5%	94.6%
LCOM1	0.071	**0.850**	−0.009	−0.013	0.028
LCOM2	−0.001	**0.981**	−0.002	−0.070	0.116
LCOM3	−0.054	**0.894**	−0.166	−0.047	0.345
LCOM4	−0.257	0.399	−0.196	−0.099	**0.851**
Co	**0.757**	−0.092	0.313	0.134	−0.216
LCOM5	−0.502	0.002	**−0.819**	−0.134	0.171
Coh	0.396	−0.144	**0.878**	0.144	−0.107
LCC	**0.925**	0.022	0.299	0.141	−0.118
TCC	**0.935**	0.025	0.303	0.114	−0.097
CBMC	0.191	−0.079	0.155	**0.963**	−0.073

A.3.1.4 Conclusion et travaux futurs

Comme conclusion, les auteurs rappellent ce qu'ils ont pu réaliser et le résultat de l'étude des métriques de cohésion existantes et de la CBMC, qui a appuyé l'efficacité de cette dernière autant que mesure de cohésion, qui, elle adopte comme critère la connectivité entre les membres d'une classe et considère les méthodes spéciales, comme méthodes ne devront pas affecter la cohésion, et qui par cela fait ressortir un nouvel aspect de propriétés et caractéristiques de classes, négligées par les métriques existantes.

Parmi les travaux futurs suggérés par les auteurs :

- Développement d'heuristiques plus appropriées avec l'utilisation d'information plus précise sur des méthodes (comme la dépendance de données entre les variables d'instance), pour une identification plus exacte des méthodes d'accès et de délégation.

- Recherche d'autres types de méthodes spéciales qui ne doivent pas affecter la cohésion d'une classe.

- Une étude empirique de la relation qui peut exister entre la métrique CBMC et les facteurs de qualité externe, telles que la maintenabilité et la facilité de réutilisation.

A.3.1.5 Références

Les auteurs ont utilisé 30 références, une seule a été analysée dans notre étude, uen seule a été analysée, il s'agit de l'article de Chidamber et Kemerer (1994). Et parmi ces références aucune ne date des années 2000.

A.3.2 Allen et Khoshgoftaar (2001) « Mesure du couplage et de la cohésion des modules logiciels : Une approche basée sur la théorie de l'information »

A.3.2.1 Introduction

Un logiciel, dont les modules ont un degré de couplage faible et de cohésion élevé a une complexité faible et donc il est de bonne qualité. Ce principe formel et logique, reste difficile à vérifier sans que les praticiens aient recours au processus de mesurage de certains attributs de qualité assez révélateurs. Ce processus exécuté dans une étape précoce de développement apporte énormément à la qualité.

La théorie de l'information est un domaine prometteur en ce qui concerne les mesures logicielles, du fait qu'on en conçoit des métriques utiles basées sur cette approche. Comme dans leur précédent article (Allen et Khoshgoftaar, 1999) déjà synthétisé, les auteurs se concentrent sur l'information représentée dans un tableau d'objet-prédicat. Cette information

a été mesurée au niveau système mais dans le présent travail elle est mesurée au niveau module.

A.3.1.1 Travaux connexes

Pour représenter un système, les auteurs utilisent une représentation similaire à celle présentée dans leur précédent article, sauf que comme nous le voyons dans la figure ci-dessous, les carreaux m1, m2, m3,...etc. ne représentent plus les sous-systèmes du système modulaire comme dans c'était le cas dans leur ancien article, mais plutôt des modules (sous-système = module).

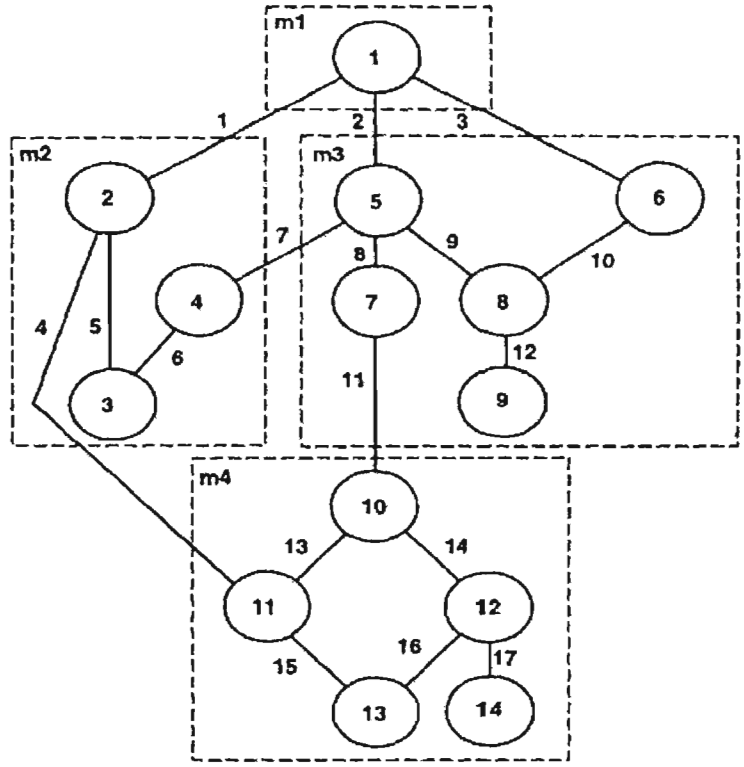


Figure A.47 Exemple de système modulaire

A.3.1.2 Protocole de mesure et Instrument de mesure

Les auteurs reprennent la définition de système modulaire de Morand, Briasca et Basili (1996) :

« *Un système modulaire est un cas spécial du système logiciel représenté par un graphe S, qui a n nœuds partitionnés dans des modules mk, k=1,..., n_M* ».

Les définitions du graphe d'arêtes inter-modules S* et du graphe d'arêtes intra-modules S0 restent les mêmes, en supposant que sous-système est synonyme de module. Et de même ces graphes sont transformés en des tables nœuds[101] * arêtes.

Les mesures que les auteurs proposent dans cet article, ont les propriétés de définition de la cohésion et du couplage dans le contexte de graphe de système modulaire, de Briand, Morasca et Basili (1996), qui ont été présentées dans le précédent article. Le protocole de mesure et instrument de mesure qui ont été adoptés par les auteurs dans leur ancien article sont les mêmes utilisés dans ce présent article en présentant les mêmes justifications.

Après, ils passent aux définitions des trois mesures.

A.3.1.3 Couplage de module

Couplage inter-modules : « *Soit MS un système modulaire, S* son graphe d'arêtes inter-modules, et mk un module, le couplage du module est défini comme suit :* »

Couplage (mk|MS) = $\sum$ I(S*i) i□mk

Avec :

I(Si) : la longueur de description minimale estimée de Si

---

[101] Les auteurs désignent par nœud un « *objet simple* ».

A.3.1.4 Couplage intra-module du module

Couplage intra-module : « *Soit MS un système modulaire, S0 son graphe d'arêtes intra-modules, et mk un module, le couplage intra-modules du module est défini comme suit :* »

$$\text{Couplage Intra-modules (mk|MS)} = \sum I(S0i) \qquad i \square mk$$

A.3.1.5 Cohésion du module

Cohésion du module: « *Soit un module mk avec nk sommets dans un système modulaire MS, sa cohésion est définie comme :* »

$$\text{Cohésion (mk|MS)} = \text{Couplage Intra-modules (mk|MS)/ Couplage (mk|MS)}$$

$$\text{Si nk} = 1 \text{ Alors, Cohésion (mk|MS)}=0$$

A.3.1.6 Étude de cas empirique

Comme étude de cas ils présentent la version 3.2.1 d'un jeu d'aventure d'ordinateur appelé *Nethack*. Le tableau ci-dessous présente le profil du programme. Ils analysent la version de base et la version courante du logiciel.

Tableau A.34 Profil du système

Name	Nethack	
Application	Adventure game	
User interface	X-Windows	
Language	ANSI C	
Operating System	UNIX	
	Baseline	Current
Files	107	111
Functions	1685	2008

Ils considèrent un fichier source comme un module, qui contient plusieurs fonctions C. En utilisant un outil créé par Jordan (1997) —qui dérive le graphe d'appel au niveau des

fonctions à partir d'un programme source C, ils représentent le graphe d'appel (call graph), dont les nœuds représentent les fonctions C et les arêtes les appels possibles entre une paire de fonctions.

À partir des calculs du couplage et de la cohésion de chaque fichier source, ils font ressortir le tableau ci-dessous qui représente les statistiques des résultats obtenus.

Tableau A.35 Résumé des statistiques

	Intermodule Coupling		Cohesion	
	Baseline	Current	Baseline	Current
Files	107	111	107	111
Average	3673	5183	0.183	0.198
Std Dev	4490	5967	0.155	0.117
Maximum	33113	43746	1.000	1.000
Median	2332	3361	0.145	0.162
Minimum	0	0	0.000	0.000

Ils ont remarqué que :

- Le couplage et la cohésion pour la plupart des fichiers source se sont amplifiés de la version de base à la version courante,
- Le couplage moyen dépasse de loin la médiane, et
- La déviation standard du couplage est plus grande que la moyenne de ce dernier.

Le nombre d'arêtes inter-modules, appelé IntermoduleEdges(mk), du module est une simple mesure conforme aux propriétés du couplage de Briand, Morasca et Basili (1996), et le rapport nombre d'arêtes intra-modules du module, et nombre d'arêtes intra-modules du

module complet [102], est aussi une simple mesure, IntramoduleRatio(mk), qui est conforme à la définition de cohésion de module donnée par Briand, Morasca et Basili (1996).

IntramoduleRatio(mk)= IntramdouleEdges(MK)/IntramoduleEdges(mk(nk))

En comparant le couplage et la cohésion des graphes d'appel basés sur la théorie de l'information avec ceux des mesures basées sur le comptage de l'ensemble de fichiers sources étudiés, les auteurs remarquent que, « *le couplage(mk|MS) et la cohésion(mk|MS) sont respectivement en corrélation avec IntermoduleEdges(mk) et IntramoduleEdgesRatio(mk)* ».

Donc, les mesures basées sur la théorie de l'information sont beaucoup plus précises que celles basées sur le comptage, et que ces dernières ne décèlent pas certains modèles de connexion que la théorie de l'information détecte. Pour illustrer cette différence, les auteurs présentent un ensemble de fichiers source de *Nethack* qui ont presque le même nombre d'arêtes inter-modules, on présente dans la figure ci-dessous deux exemples de ces fichiers sources avec leurs graphes inter-modules.

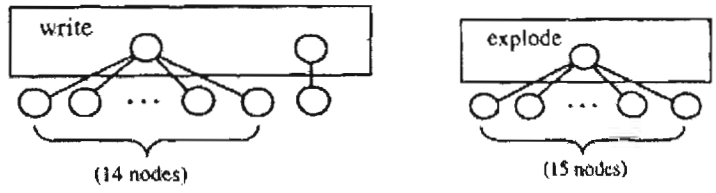


Figure A.48 Exemple de couplages de graphes d'appel

IntermoduleEdges(Write|MS) =15arêtes *IntermoduleEdges(explode|MS)* =15arêtes

Couplage (Write|MS) = 539.5 bits *Couplage(explode|MS)* = 529.3 bits

[102] Un module complet est un module dont chaque nœud et lié à tous les nœuds du même module.

Les graphes des fonctions *write* et *explode* ont le même nombre d'arêtes, mais des couplages plus ou moins différents.

A.3.1.7 Travaux futurs

Parmi les travaux futurs que les auteurs suggèrent, nous rapportons ci-dessous les plus importants :

- Une étude comparative des métriques basées sur le comptage et celles basées sur la théorie de l'information.
- Validation de l'utilité de la cohésion et du couplage de module dans un contexte de modèles de prévision de qualité.
- Adopter la théorie de l'information sur d'autres familles de métriques.

A.3.1.8 Réféfences

Les auteurs ont utilise trente-cinq (35) références :

- Deux livres, les deux datent des années 1990, celui de Pressman (1992) et un autre qui traite de la théorie de l'information aussi des années 1990.
- Trois thèses : la thèse d'Allen, dont le sujet est la théorie de l'information et mesure de logiciel, la deuxième thèse parle des mesures de la cohésion et du couplage du logiciel et la dernière est celle de Jordan dans laquelle il a proposé deux nouveaux outils propres aux métriques logicielles, et parmi ces outils, un a été employé par les auteurs dans l'étude de cas empirique.
- Vingt-huit articles,
    - Un des années 1940, le célèbre article de Shannon et Weaver qui présente leur théorie de la communication.
    - Un des années 1950, il s'agit de l'article de Miller, concernant le nombre magique sept.
    - Un des années 1970,
    - Trois des années 1980, qui portent tous sur la complexité, son évaluation, ses mesures, et son impact sur la fiabilité du logiciel, l'article de Weyuker (1988) a été résumé dans l'Appendice C.

- Vingt-deux des années 1990, dont deux été synthétisés dans le même chapitre, il s'agit de l'article précédent des mêmes auteurs mis en référence dans la section Introduction, et celui de Bieman et Kang (1998).

- Deux références dont la nature nous est inconnue

A.3.3 Darcy et Kemerer (2002) « Complexité de logiciel : Vers une théorie unifiée du couplage et de cohésion »

A.3.3.1 Introduction

Pour souligner le besoin de recherches théoriques sur les mesures de la complexité, dans l'introduction les auteurs commencent par présenter huit (8) citations tirées de livres et revues allant de 1984 à 2000. Ils mentionnent ensuite que durant les années 1990 plus de cent (100) métriques de complexité ont été conçues, et plus de cinq-cents (500) articles traitant de ce sujet ont été publiés.

A.3.3.2 Fondements théoriques

Dans cette partie les auteurs présentent dans un premier lieu le modèle de Wood (1986) de la complexité de la tâche, ensuite les concepts de traitement de l'information en général et en particulier, le traitement de l'information lié au génie logiciel. Cette partie termine avec une présentation des métriques pour la cohésion et pour le couplage et avec un l'examen de certaines recherches empiriques.

A.3.3.2.1 Le modèle de complexité de la tâche de Wood (1986)

Les auteurs ont choisi le modèle de Wood entre autres parce qu'il permet de « *représenter la complexité de la tâche de manière indépendante du niveau cognitif nécessaire pour l'exécuter* ». Ils reprennent les définitions de Wood des composants des tâches :

Produit : « *entités créées ou produites par les comportements et pouvant être observées et décrites indépendamment des comportements ou des actes qui les produisent* ».

Acte : « *une suite de comportements avec des but ou des directions distinctes* ».

Indice : « *un renseignement concernant les attributs des objets qui causent les stimuli : attributs sur lesquels un individu peut baser les jugements qu'il doit porter pendant l'exécution de la tâche* ».

La figure ci-dessous montre les relations entre ces trois composants.

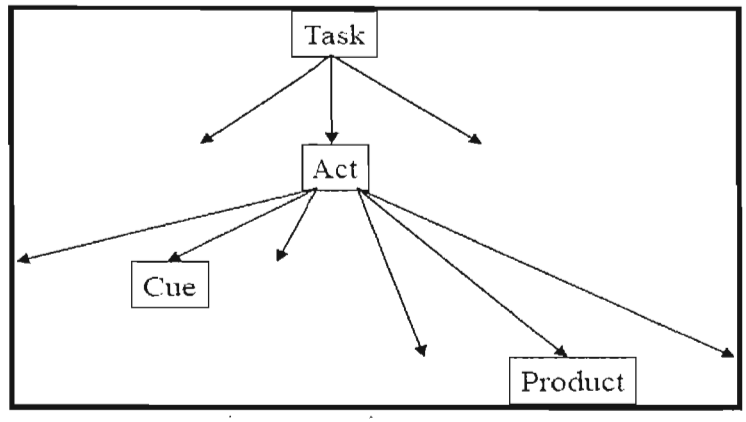


Figure A.49 Les composants des tâches

« *En utilisant les trois atomes des produits, des actes et des indices, trois sources de tâche de la complexité ont été définies* :» complexité du composant, complexité de coordination et complexité dynamique. Dans ce qui suit, nous allons les présenter :

Complexité du composant **:** Il s'agit de « *la fonction du nombre d'opérandes ou jetons de données qui doivent être traités lors de l'exécution de tous les actes requis pour l'accomplissement de la tâche* ». La complexité du composant (TC_1) est représentée par l'équation suivante :

$$TC_1 = \sum_{j=0}^{p} \sum_{i=1}^{t} W_{ij}$$ Équation (1)

Où :

- P est le nombre de tâches secondaires dans une tâche.
- L est le nombre d'actes dans une tâche secondaire.
- Wij est le nombre de sélections de l'information à traiter pour l'acte i de la tâche secondaire j.

La complexité du composant s'accroît avec l'augmentation du nombre d'actes distincts.

Complexité de coordination : Cette complexité « *couvre la nature des relations entre les entrées de la tâche et ses produits* ». La complexité de coordination (TC_2) est représentée par l'équation suivante, à ne considérer que les relations distinctes :

$$TC_2 = \sum_{i=1}^{n} R_i$$

Où :

- N : le nombre d'actes dans la tâche, et
- Ri : le nombre de relations de priorité entre l'acte i et tous les autres actes dans la tâche.

La forme, la force et l'ordonnancement sont tous des aspects de la complexité coordonnatrice.

Complexité dynamique : Ce sont les « *changements dans les états du monde qui ont un effet sur les relations entre les tâches et les produits* ». La complexité dynamique (TC_3) est représentée par l'équation suivante :

$$TC_3 = \sum_{t=1}^{m} \left| TC_{1(t-1)} - TC_{1(t)} \right| + \left| TC_{2(t+1)} - TC_{2(t)} \right|$$

Équation (3)

Où M, est le nombre de périodes de temps pendant lesquelles la tâche est mesurée.

Enfin, les auteurs présentent l'équation (4) qui correspond à la formule de calcul de la tâche de la complexité totale (TCt), qui est une fonction de combinaison linéaire des trois types de complexité, chacun exprimé par une unité normalisée. Dans cette équation, l'unité de la complexité dynamique contribue plus que celle de la complexité coordonatrice, qui à son tour contribue plus que la complexité du composant (ce qui est exprimé par : $\alpha < \beta < \gamma$).

$$TC_t = \alpha TC_1^{\ s} + \beta TC_2^{\ s} + \gamma TC_3^{\ s}$$ Équation (4)

Où :

- TC1s : complexité du composant mesurée dans les unités normalisées.
- TC2s : complexité de coordination mesurée dans les unités normalisées.
- TC3s : complexité dynamique mesurée dans les unités normalisées.

A.3.1.1.1 La perspective du traitement de l'information sur la cognition

De nos jours, la majorité des psychologues analysent les phénomènes cognitifs en faisant des analogies avec les ordinateurs. À ce propos, on parle donc de mémoire à court et à long terme :

- Mémoire à court terme (MCT) : dans laquelle nous devons avoir toutes les instructions avant de réfléchir. Ce type de mémoire est petit par rapport au deuxième type, sa capacité qui a été déterminées par Miller (1956) est limitée à 7±2 articles.

- Mémoire à long terme (MLT) : dans laquelle un traitement de processus ou de donnée est largement lent par rapport à un traitement effectué par la STM.

Ce qui diffère entre l'ordinateur et le cerveau humain, c'est que dans ce dernier une unité de stockage n'est pas aussi homogène que l'est un octet dans un ordinateur. Les unités de stockage pour un cerveau sont appelées des « fragments ». Le stockage de ces fragments veut aussi dire que nous comprenons ce qui se passe à l'intérieur de ces fragments.

A.3.3.3 C&C : un peu d'histoire

Dans la première sous-section qui concerne les mesures de C&C dans la programmation procédurale, les auteurs présentent l'histoire bibliographique de ces deux concepts comme le montre la figure ci-dessous.

Selon les auteurs, la source de C&C revient à l'article de Stevens, Myers et Constantine (1994), à partir duquel chacun des auteurs a rédigé un livre sur la programmation structurée et dans lesquels nous trouvons un chapitre sur le couplage et un autre qui traite de la cohésion. Bien que critiqués à cause du manque de fondements théoriques et de rigueur, ils restent des points de départ incontournables des études de C&C.

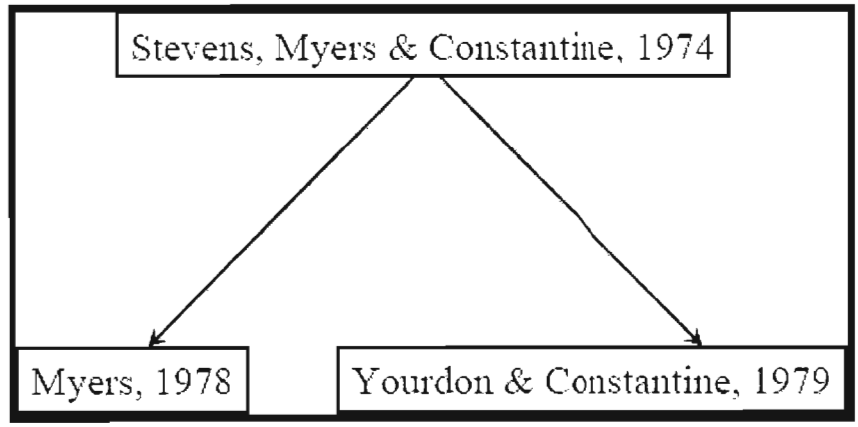


Figure A.50 Travaux majeurs sur C&C

Les auteurs donnent les définitions C&C et leurs catégorisations comme ils apparaissent dans les travaux originaux (Chapitre 3).

Ces trois travaux forment la base de la plupart des recherches faites sur les métriques de C&C pour la programmation procédurale. Le tableau ci-dessous présente la liste chronologique des travaux majeurs dans lesquels des mesures de C&C ont été développées. À partir de ce tableau, on peut clairement remarquer que la majorité de ces travaux ont une nature observationnelle ou conceptuelle, comprenant très peu de validation empirique des mesures. En gros, le tableau contient : deux travaux n'incluant aucune partie empirique, six autres travaux qui analysent en moyenne sept (7) systèmes commerciaux moyens et quelques petits exemples.

Tableau A.36 Métriques de C&C pour la programmation procédurale

Cite	Measures	Empirical Data
[73]	4 coupling measures (data bindings)	2 medium commercial systems
[90]	5 coupling measures	None reported
[91]	Combined measure including 1 coupling and 1 cohesion measure	1 medium Unix utility
[92]	1 cohesion and 5 coupling measures	Textbook system
[93]	8 measures for both coupling and cohesion	1 medium system
[72]	11 measures of coupling	2 small and 3 medium systems
[71]	7 cohesion measures (for Table 2 levels)	None reported
[19]	1 measure for functional cohesion	5 sample procedures were analyzed

Avec l'arrivée de la programmation OO et son adoption aussi lente que prévue, le besoin de nouvelles mesures de C&C autres que celles existantes, qui sont spécifiques à la programmation procédurale, se fait sentir. En effet, la programmation procédurale est caractérisée par une terminologie, des notions et des principes qui sont différents de la programmation OO. Pour remédier à ce problème, Chidamber et Kemerer (1994) conçoivent une suite de six (6) métriques pour la programmation OO appelée suite de CK, et qui est fondée sur une base théorique solide. Cette suite comprend, une métrique de cohésion (LCOM) et deux de couplage (CBO et RFC).

Après la production de la suite CK, deux revues des métriques de l'OO ont été réalisées et ont listé treize (13) métriques de la cohésion et trente (30) pour le couplage, basées sur la suite de CK. Cette dernière a constitué un vrai départ d'apparition de métriques de rigueur. En outre, cette suite a été intensivement et empiriquement validée.

Par ailleurs, une étude empirique sur un système commercial montre que parmi toutes les mesures existantes, les métriques originales de CK : CBO et LCOM, sont en parfaite corrélation avec la tendance de l'erreur d'une classe.

Avec la programmation OO, les recherches traitant C&C ont pris de nouvelles directions, comme leur utilisation en tant que prédicateurs de la qualité de la conception ou en tant que mesures de la complexité structurée (un faible couplage et une forte cohésion veut dire une bonne restructuration des modules du logiciel).

A.3.3.4 Modèle de recherche : Couplage et Cohésion comme mesures de complexité

D'abord, il serait essentiel d'effectuer une correspondance entre la terminologie générale du modèle de Wood (1986) et celle du domaine de la complexité du logiciel ; une sorte de parallélisme terminologique. Le tableau ci-dessous présente cela.

Tableau A.37 Correspondance terminologique entre le modèle de Wood et la complexité logicielle

Modèle de Wood		**Complexité logicielle**	
Composants essentiels :	Acte	Unité de programme	
	Produit	Sortie d'une unité de programme	
	Repérage d'information	**Terminologie de Bieman :**	Jetons de donnée
		Terminologie de Halstead :	opérandes[103]
Tâches de complexité :	Complexité du composant	Cohésion	
	Complexité coordonnatrice	Couplage	
	Complexité dynamique	Changement de la cohésion et du couplage et de la cohésion avec le temps	

En utilisant les termes de Wood (1986), la cohésion peut être définie comme « *la mesure de « l'union » d'une « unité de programme »*[104] *ou d'un acte* ». Donc, on voit bien l'équivalence entre la complexité du composant et la cohésion au niveau d'une unité de programme[105].

Le couplage est vu comme « la mesure de la « parenté » » d'une unité de programme à d'autres unités. Cette parenté est équivalente à la priorité au niveau d'une unité de programme[106], puisque l'accomplissement d'un acte exige la résolution de tous les liens pour

---

[103] Les opérateurs pourraient être considérés comme des repérages de l'information. Cependant, ils font partie de l'acte lui-même, puisqu'ils contribuent en la production des sorties mais ne constituent guère des sorties.

[104] Les auteurs choisissent le terme « unité de programme » pour se référer à tout ce qui peut être synonyme du mot module, comme : classe, procédure, fonction, structures de données avec une/plusieurs procédures/fonctions.

[105] Le modèle de Wood permet aussi de considérer la cohésion du programme de tout le système.

[106] L'analyse de Wood permet encore de considérer tout le programme.

cette unité de programme. » Donc, la complexité coordonatrice est équivalente au couplage en GL.

La complexité dynamique n'est pas considérée car elle constitue les changements des complexités du composant et coordonatrice à travers le cycle de vie du logiciel.

A.3.3.5 Complexité du logiciel : Modèle de recherche

Revenant aux deux questions de recherche de l'article. La première question tournait autour du choix d'un ensemble de mesures de la complexité logicielle, dans laquelle on se trouve face à plusieurs passages. Parmi ces passages, trois sont frappants :

- La motivation pour la question : Les chercheurs et praticiens ont besoin de focaliser sur un sous-ensemble particulier de mesures, pour mieux procéder à l'évaluation, la prédiction et le contrôle de la complexité logicielle.

- Comment déterminer l'adhésion du sous-ensemble de mesures choisi? Au début on proclamait la nécessité d'une seule mesure, mais après il s'est avéré qu'on avait besoin de plus d'une mesure, mais pas nécessairement toutes celles existantes.

- Comment choisir ce sous-ensemble de mesures? Si on considère les activités du logiciel (comme la conception, la programmation et la maintenance) comme des tâches ayant besoin d'être analysées par des méthodes et des modèles, tel que le modèle de Wood (1986) et en se basant sur les résultats de la sous-section précédente, le couple C&C constitue un ensemble suffisant de métriques pour bien cerner la variabilité de la structure du logiciel. Car, comme on l'a déjà vu, le couple est équivalent aux sources de tâches de la complexité coordonnatrice et du composant.

Répondons maintenant à la deuxième question. Si le couplage augmente, l'analyse de modules typiques est ajoutée aux fragments qui représentent les unités de stockage d'informations utiles pour la compréhension d'un module donnée par le programmeur. Si les parties ajoutées sont faibles en cohésion, l'effort pour la compréhension du module est encore plus important. Ceci amène les auteurs à faire la proposition suivante :

P1 : Pour des modules fortement couplés, la performance de compréhension diminue.

Les mesures de C&C sont habituellement examinées séparément, mais leurs effets sur la complexité sont liés. Actuellement l'effet de la cohésion est considéré comme un effet commun au couplage. Et la proposition suivante est faite :

P2 : La compréhension des modules fortement couplés est améliorée lorsque l'on est en présence d'une forte cohésion.

En conséquence, et en réponse à la deuxième question de recherche, la relation entre C&C est interactive et c'est le couplage qui se présente comme le conducteur de la performance de compréhension d'un module. La figure ci-dessous présente le modèle proposé qui relate ces relations.

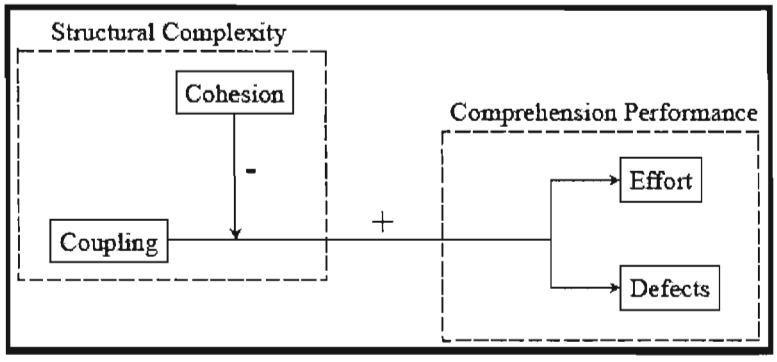


Figure A.51 Modèle de recherche

Ce modèle est présenté indépendamment du paradigme et du langage de programmation. D'où vient la proposition suivante :

P3 : L'impact du couple C&C sur la compréhension d'un module n'est nullement influencé par le langage de programmation.

Il existe plusieurs autres façons de reconfiguration du modèle présenté dans la figure 9.13, dans laquelle on a supposé le couplage comme effet principal avec la cohésion qui

modère ce rapport. Mais, réellement on ne sait pas qui constitue le rapport principal et qui est le modérateur d'entre C&C.

A.3.3.6 Conclusions et travaux futurs

Le couple C&C a été historiquement bien analysé et développé, mais il n'a jamais été clair comment il est important et essentiel comme indicateur de la complexité logicielle. Il a été « *conventionnellement considéré comme un couple de concepts indépendants, et comme quelque chose sous le défi de l'analyse et le défi du modèle proposé* ». D'autre part, ce couple est bien doté de mesures et d'opérationalisation[107].

Les auteurs pensent que les futures recherches peuvent être concentrées sur la confirmation et le raffinage des mesures et des modèles de C&C.

A.3.3.7 Références :

Dans cet article, les auteurs ont utilisé 99 références :

- Huit (8) livres qui datent des années 90, 80 et 70.
- Quatre-vingt (83) trois articles.
    - Deux (2) des années 50, dont un (1) traite le nombre magique 7.
    - Un (1) des années 60.
    - Sept (7) des années 1970, qui parlent en général de la programmation structurée.
    - Dix-huit (18) des années 1980, dont quatre (4) traitent la complexité, deux (2) la psychologie de la programmation ou la psychologie cognitive, les autres traitent soit de la maintenance du logiciel ou la programmation structurée, parmi ces articles un a été analysé, celui de Weyuker (1988) dans l'Appendice C.
    - Quarante-neuf (49) des années 1990, dont seulement neuf (9) sont liés directement à la cohésion et au couplage, les autres traitent soit, la

[107] « *Mise à l'épreuve pratique des concepts théoriques* » (OQLF, 2008)

complexité, la maintenance, la philosophie des mesures, l'approche OO, ou la psychologie de la programmation. Et parmi ces articles deux (2) ont été analysés en détails celui de Chidamber et Kemerer (1994) et celui de Lakhotia (1993).

- Huit (8) des années 2000, dont seulement un (1) qui parle directement de la cohésion.

A.3.4 Meyers et Binkley (2004) « Métriques de cohésion basées sur le tranchage et intervention sur le logiciel »

A.3.4.1 Introduction

Les recherches les plus importantes concernant les métriques de cohésion sont celles qui se fondent sur l'approche de tranchage élaborées par Bieman, Ott. et Thuss (Ott et Thuss, 1993) (Bieman et Ott, 1994). Ces métriques, à cause du manque d'outils souffrent de plusieurs limitations empiriques : comme la petite taille et le nombre limité de programmes étudiés ou les limitations des outils comme, par exemple, leur incapacité à traiter les pointeurs. Mais l'évolution qu'a connue la technologie de tranchage et l'apparition d'outils plus sophistiqués, permettent aujourd'hui de réaliser des études empiriques plus importantes en termes de taille et de nombre de programmes.

Les auteurs cherchent à savoir qu'elles sont les valeurs des métriques pour un cas « normal » et pour un cas nécessitant une intervention sur le logiciel.

L'étude empirique réalisée par les auteurs de cet article, analyse 22 651 procédures[108] de 63 programmes (de 500 à 50 000 lignes de code), ce qui donne en tout, plus de 1.1 million de lignes de code et 2, 067,016 tranches. Cinq métriques (*Tightness*, *MinCoverage*, *Coverage*, *MaxCoverage* et *Overlap*) sont appliquées aux 63 programmes. Pour la description des métriques voir le Chapitre 5.

---

[108] Pour les auteurs de cet article, « procédure » et synonyme de « module ».

Les auteurs considèrent que cet article apporte trois contributions importantes à l'étude de la qualité des programmes à l'aide du tranchage :

1. Les mesures de cohésion fondées sur le tranchage, à cause de leur corrélation avec la qualité, peuvent être employées « *pour guider et mesurer les efforts d'intervention sur le logiciel* ».

2. Les valeurs de base pour les cinq (5) métriques considérées « *sont utiles pour identifier les modules dégradés* ».

3. Une comparaison des métriques montre « *quelles métriques sont fortement corrélées et lesquelles donnent des visions différentes des programmes* ».

Les auteurs, avant de décrire leurs expériences, présentent les fondements théoriques du tranchage que nous ne résumons pas car nous les avons déjà vus dans le Chapitre 5. Trois parties sont considérées dans leur étude :

A.3.4.2 Valeurs de base des métriques

Le tableau ci-dessous, présente les moyennes des cinq métriques pour les 22,651 modules étudiés. À partir des données de ce tableau, les auteurs font ressortir les observations suivantes :

- La grande dispersion d'*Overlap*, montre que cette dernière est une métrique très sensible.

- Réciproquement, *MaxCoverage* est la métrique la moins sensible parmi toutes celles étudiées.

À partir de ces deux observations, les auteurs tirent la conclusion suivante : « *les valeurs fournies par MaxCoverage sont plus significatives que celles d'Overlap* ».

Tableau A.38 Moyennes de métriques pour les 22,651 modules

	SDG vertices	number of slices	$\|SL_{int}\|$	*Tightness*	*Min-Coverage*	*Coverage*	*Max-Coverage*	*Overlap*
average	897.35	84.65	276.17	0.3006	0.3387	0.5402	0.6453	0.5436
standard deviation	2,398.45	139.91	1,100.46	0.2556	0.2570	0.1693	0.1547	0.3454
confidence interval	31.23	1.82	14.33	0.0033	0.0033	0.0022	0.0020	0.0045

Les auteurs présentent par la suite les résultats des mesures, mais cette fois-ci pour les programmes non plus pour les procédures. Les moyennes des valeurs des mesures sont présentées par le graphique de la figure ci-dessous. À partir des données de cette figure, les auteurs tirent les considérations suivantes :

- À cause de la grande variation des moyennes des métriques *Tightness*, *MinCoverage* et *Overlap*, ces dernières peuvent fournir une bonne discrimination de la cohésion.
- Toutes les mesures augmentent quand *Tightness* augmente.
- La sensibilité de *MinCoverage* aux changements de *Tightness* est très faible.
- *Coverage*, *MaxCoverage* et *Overlap*, sont sensibles aux changements de *Tightness*.
- *Overlap* a un comportement particulier surtout si on la compare à *Coverage* et *MaxCoverage*.

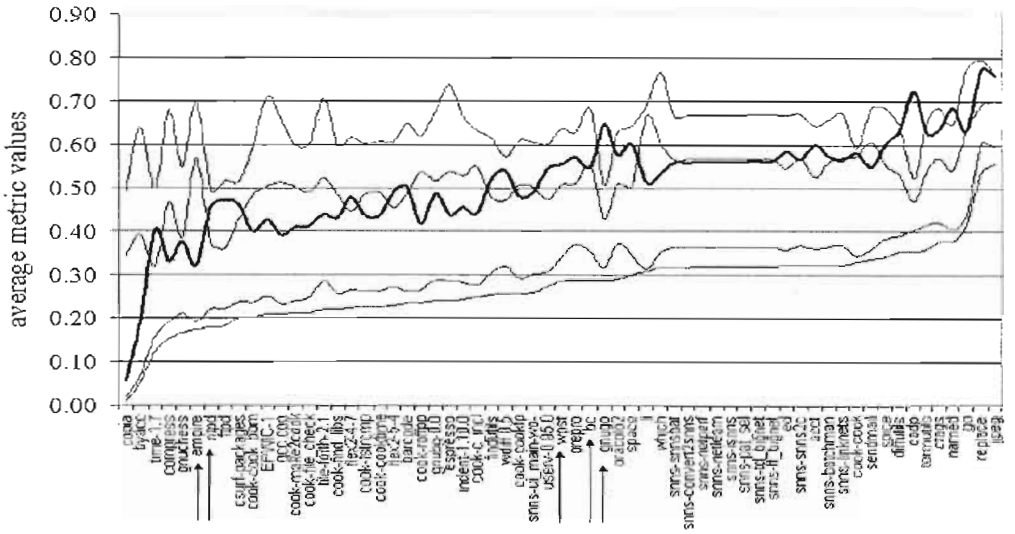


Figure A.52 Les moyennes des métriques pour chaque programme, triées[109] par la métrique *Tightness* [110]

A.3.4.3 Études temporelles

Deux études temporelles sont présentées pour analyser la sensibilité des métriques par rapport aux différentes versions d'un programme. Les programmes choisis sont : *barcode* (9 versions mineures) et *gnugo* (7 versions majeures).

Les résultats de ces deux études sont présentés par les figures A.53 et A.54. Les figures A.53.a et A.53.b, présentent l'évolution (taille-nombre de tranches-|SLint|) de *barcode* et *gnugo* respectivement. Les figures A.54.a et A.54.b présentent les moyennes des cinq métriques pour quelques versions des programmes *barcode* et *gnugo*.

---

[109] « Ce tri aide à visualiser certaines caractéristiques. »

[110] La ligne en gras représente la métrique *Overlap*, alors que les lignes restantes représentent du bas en haut, *Tightness*, *MinCoverage*, *Coverage* et *MaxCoverage*.

La figure A.53.a montre une grande divergence entre la courbe de l'évolution du programme et de |SLint|[111], ce qui implique une détérioration de la cohésion du programme. C'est-à-dire que la maintenance affaiblit la qualité de ce programme.

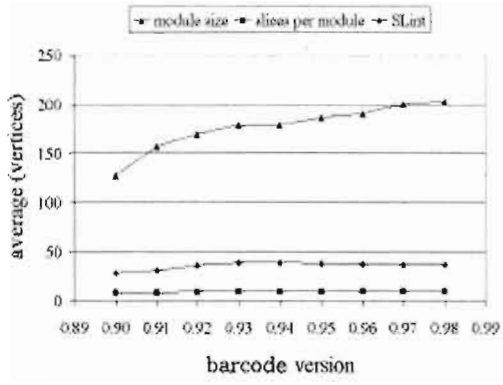
La figure A.53.b montre que toutes les métriques à l'exception d'*Overlap*, indiquent une forte cohésion dans la première ou deuxième version du programme *barcode*. On remarque aussi à partir de cette figure, que l'Overlap de *barcode*, a une tendance très différente par rapport à toutes les autres métriques. Bref, les résultats des cinq métriques prouvent que la cohésion du programme diminue.

D'après la figure A.54.a on remarque que la version 3.0 de *gnugo* est la version dans laquelle on remarque une convergence assez importante entre la taille du programme et |SLint|.

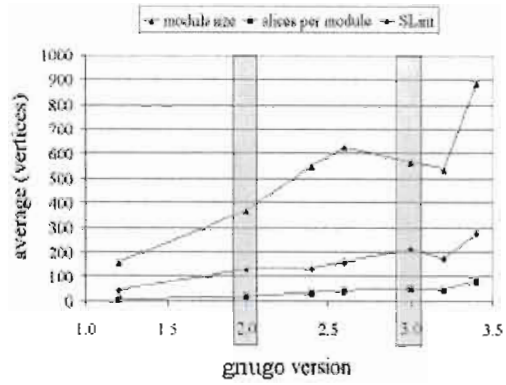
La figure A.54.B montre que toutes les métriques, après avoir atteint leurs valeurs maximales à la première ou deuxième version du programme *gnugo*, régressent. Elle montre aussi que les révisions majeures du programme (versions 2.0 et 3.0), sont très intéressantes, car il y a une amélioration significative de la cohésion pour toutes les métriques sauf *Overlap*.

---

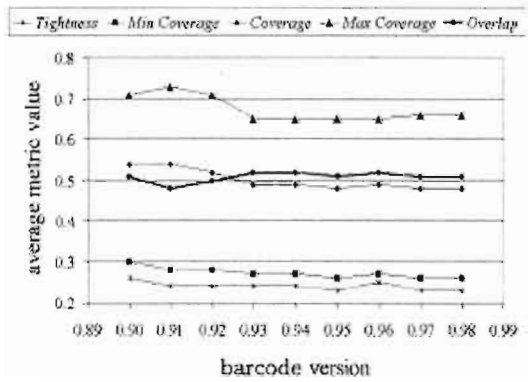
[111] |SLint| représente le nombre d'instructions communes entre toutes les tranches du même programme.



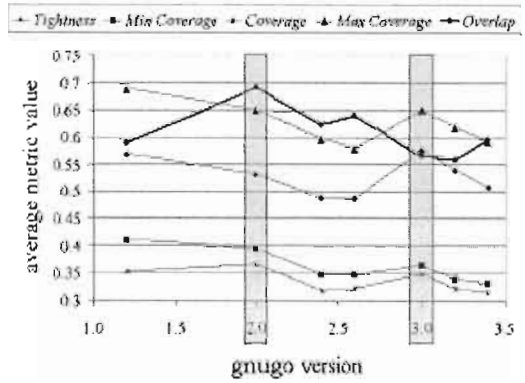
(a) Growth



(a) Growth



(b) Metric Values



(b) Metric Values

Figure A.53 Étude longitudinale de *barcode*

Figure A.54 Étude longitudinale de *gnugo*

A.3.4.4 Comparaison des métriques

Enfin, les auteurs procèdent à une comparaison quantitative et autre qualitative des cinq métriques étudiées.

Dans la comparaison quantitative, les auteurs appliquent le test de *Pearson* pour les valeurs produites par chaque paire de métriques.

Le tableau ci-dessous, présente le coefficient de corrélation de *Pearson* de chaque paire de métriques R, et R^2 qui désigne le pourcentage de dépendance. Les auteurs sont arrivés à déceler trois relations assez importantes :

- Une corrélation forte entre *MinCoverage*, *Tightness* et *Overlap*.
- *MaxCoverage* est la métrique la plus faiblement corrélée avec toutes les autres.
- *Overlap* est la métrique la plus variable, car elle montre une corrélation avec *MinCoverage* et *Tightness* et d'autre part, ne montre aucune relation linéaire avec *Coverage* et *MaxCoverage*.

Tableau A.39 Les valeurs de R et R^2 pour chaque paire de métriques

x	y	R	R^2
MinCoverage	Tightness	0.977	0.955
Overlap	Tightness	0.916	0.839
Overlap	MinCoverage	0.896	0.803
MaxCoverage	Coverage	0.793	0.628
Coverage	Tightness	0.625	0.391
Coverage	MinCoverage	0.602	0.362
Tightness	MaxCoverage	0.379	0.144
MinCoverage	MaxCoverage	0.367	0.135
Coverage	Overlap	0.356	0.126
MaxCoverage	Overlap	0.154	0.024

Les trois figures ci-dessous, représentent des exemples de comparaison qualitative des cinq métriques.

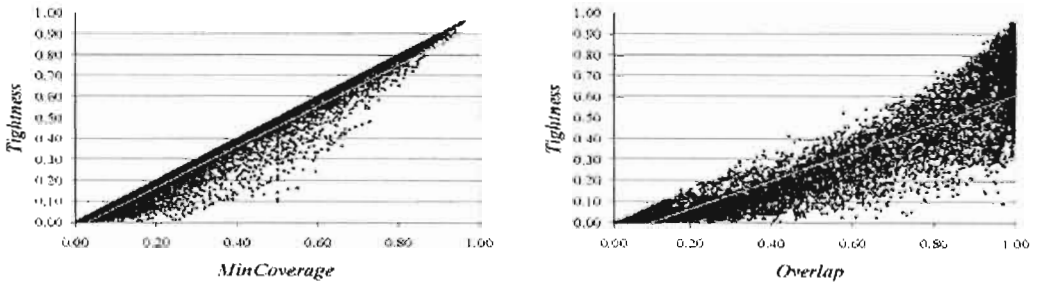


Figure A.55 Exemple de comparaison montrant une forte corrélation linéaire

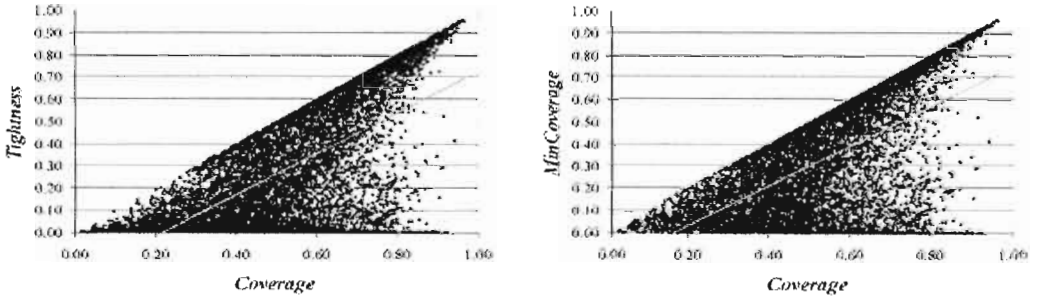


Figure A.56 Exemple de comparaison montrant une faible corrélation linéaire

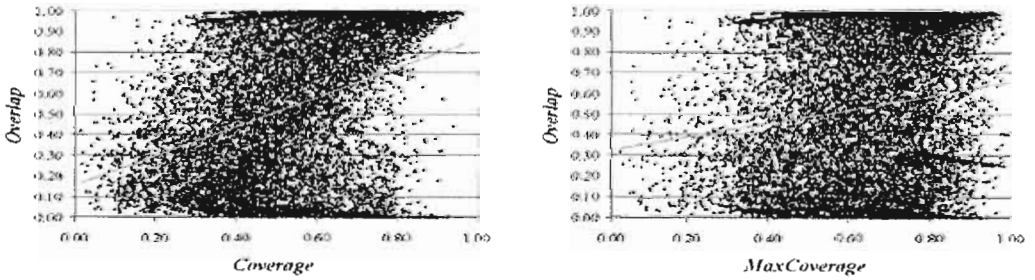


Figure A.57 Exemple de comparaison montrant une corrélation non linéaire

La Figure A.55, montre une corrélation évidente entre *MinCoverage* et *Tightness*, car le nuage des points s'accumule tout au long près de la droite x=y c.à.d. une parfaite corrélation R=1. La Figure A.57, montre clairement que les métriques *MaxCoverage* et

Overlap fournissent deux visions différentes du même programme, donc il est possible qu'elles soient utilisables mais chacune a sa façon de réagir.

A.3.4.5 Travaux futurs

Les résultats de cette étude montrent un besoin d'amélioration ou de remplacement de la définition de *Parallelism*. Et quatre types de recherches empiriques ont été suggérées :

- Étudier les métriques basées sur le tranchage.
- Considérer l'efficacité des métriques *glue-token*.
- Étudier en profondeur le couplage basé sur le tranchage.
- Réaliser une étude pour répondre à la question : « *le développement de logiciel ayant accès aux valeurs des métriques basées sur le tranchage permet-il de mieux restructurer les programmes ?* »

A.3.4.6 Références

Dans cet article, les auteurs ont utilisé 19 références :

- Trois (3) livres, dont un traite l'approche tranchage, le deuxième est celui de Yourdon et Constantine (1979) et le dernier qui date des années 90 et parle du comportement des métriques de cohésion basées sur le tranchage.
- Neuf (9) articles, dont deux (2) ont été consultées au cours de notre recherche, il s'agit des articles de Bieman et kang (1998) et de Bieman et Ott (1994), et deux (2) synthétisés c'est l'article d'Ott et Thuss (1993) et celui des mêmes auteurs publié en 1989 (Ott et Thuss, 1989).
- Six communications, dont une consulté durant notre recherche, c'est celle de Weiser (1981). Seulement deux de ces communications ont été tenues dans les années 2000.
- Un rapport technique qui traite l'approche tranchage qui date des années 1980.

A.3.5 Yang et Berrigan (2005) « Détection du couplage indirect »

L'intérêt du couplage réside dans le fait qu'il est censé avoir une influence sur la qualité du logiciel et en particulier sur la facilité de maintenance. C'est sur ce dernier attribut que se concentre l'article. Les considérations des auteurs se limitent à l'approche OO et ils emploient le terme « module » comme synonyme de « classe ».

A.3.5.1 Contexte

La définition de couplage de Yourdon et Constantine (1979), tout en ayant donné naissance à de bonnes heuristiques, est trop informelle car elle utilise des termes mal définis comme « interconnexion » et « force ». Et c'est le cas d'autres tentatives de définition de couplage, qui se basent aussi en quelque sorte sur la définition donnée par Yourdon et Constantine (1979), considérée comme « la définition originale du couplage ». Les auteurs citent aussi Fenton et Pfleeger (1997) , pour montrer comment même les mesures n'aident pas à mieux définir le couplage car « *Il n'y a pas de mesures normalisées du couplage* ».

Ensuite ils mentionnent que la majorité des métriques de couplage existantes concernent le couplage direct. Briand, Daly et Wust (1999), par exemple, ont présenté trente (30) métriques de couplage dont uniquement deux (2) mesurent des formes de couplage indirect. Briand, Daly et Wust (1999) donnent aussi une définition de couplage indirect en lien avec le couplage directe : « *Le couplage direct décrit une relation entre un ensemble d'éléments* ([...]). *Pour expliquer le couplage indirect, nous avons seulement besoin d'utiliser la fermeture transitive de cette relation* ». Les auteurs considèrent que cette définition de « couplage indirect » n'est pas utile, car d'un côté elle ne capture pas toutes les formes du couplage indirect et d'un autre elle peut indiquer que des classes sont couplées alors qu'elles ne le sont pas.

A.3.5.2 Définition du couplage

Cette partie commence par un exemple que les auteurs emploient pour montrer la faiblesse de la définition du couplage indirect de Briand, Daly et Wust (1999). Pour faciliter

la lecture nous avons agencé les deux figures de l'article l'une à côté de l'autre. La figure ci-dessous présente l'exemple.

```
class A {
  B aB = new B();
  E aE = new E();
  void doitA() {
    aB.doitB();
    aE.doitE();
  }
}

class B {
  C aC = new C();
  void doitB() {
    aC.doitC();
  }
}

class C {
  D aD;
  C() {
    aD = new D();
  }
  void doitC() {
    // no uses of aD
  }
}

class E {
  void doitE() {
    // ...
  }
}
```

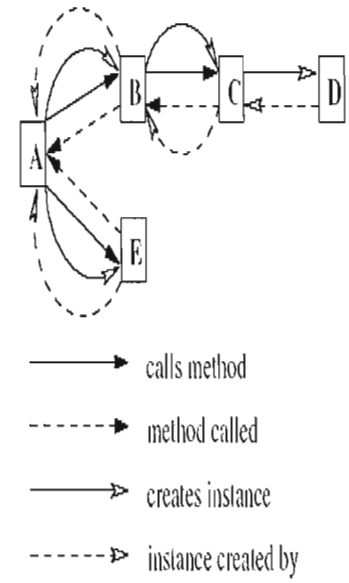


Figure A.58 Exemple pour montrer les problèmes que peut causer la définition de couplage indirect de Briand, Daly et Wust (1999)

Les auteurs considèrent quatre types de relations entre les classes : Appel de méthode et méthodes appelées d'une part et création d'une instance et instance crée d'autre part.

1) Appel de méthode : Cette relation est transitive, non symétrique, et considérée comme relation de couplage. Les classes A et B sont directement couplées par cette relation, aussi B et C, et, A et E. Tandis que A et C ne sont pas couplées.

2) Méthode appelée : On remarque que C et B sont couplées, B et A le sont aussi, et, E et A, mais A n'est pas couplée à C et D.

3) Appeler méthode ou méthode appelée : Elle est la composition des deux méthodes précédentes. Avec cette nouvelle relation, on remarque que les classes A et C sont indirectement couplées dans les deux directions.

Par ces trois relations, A est couplée à E, et couplée aussi à C, et, E et C sont indirectement couplées, mais ces deux dernières classes n'ont aucune relation entre elles, et il est considéré irraisonnable de les considérer comme couplées. Donc, il serait incorrecte «*...de définir le couplage direct comme étant la composition d'un ensemble de relations de couplage direct, car cela nous mène a considérer la plupart des modules du système comme couplés, ce qui est n'est pas très utile.*»

Dans ce qui suit, on nomme le couplage indirect résultant d'une clôture (fermeture) transitive d'une simple relation « couplage transitif simple (*Simple Transitive Coupling STC*) », et celui identifié par une clôture transitive de la composition de plus d'une relation le « couplage transitif composé (*Composite Transitive Coupling CTC*) ».

Et Donc, il faut donner une définition plus acceptable au couplage indirect, en effet les auteurs le définissent comme : « *n'importe quel couplage qui n'est pas un couplage direct* » Mais le problème avec cette définition c'est qu'elle n'est pas opérationnelle, car au fond la définition du couplage est informelle. Pour remédier à ce problème, un critère utile à appliquer pour savoir si deux classes sont couplées ou non, c'est de répondre à la question de Yourdon et Constantine (1979), « *à quel point un module doit être connu pour comprendre un autre module ?* »

A.3.5.3 Exemples de couplage indirect

Les auteurs dans cette section présentent un exemple de couplage indirect entre deux classes qui n'a pas la forme STC. Cet exemple est pris d'un simple système de gestion d'une bibliothèque appelé SLMS. La figure ci-dessous, montre une partie de la conception de SLMS. Dans laquelle, la classe SLMS maintient deux listes, une contenant les clients habituels de la bibliothèque, représentée par la classe *Patron*, et l'autre contient les livres de la bibliothèque, représentée par la classe *Book*. Les livres sont de différents types comme : référence, adulte, adolescent, enfant (représenté par *BookType*), qui détermine comment ils peuvent être empruntés, et les patrons sont de différentes catégories, comme : adulte, enfant (représenté par *PatronCategory*), qui définissent quels livres ils peuvent emprunter.

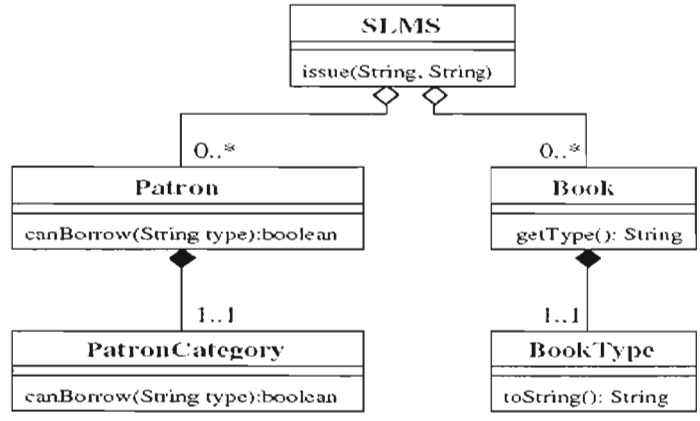


Figure A.59 Diagramme de classe d'une partie de la conception de SLMC

En vérifiant le code source de *BookType* et *PatronCategory*, on ne peut remarquer aucune relation de couplage direct entre ces deux classes. Mais pour qu'un livre puisse être emprunté par un client il doit d'abord être publié. La figure ci-dessous, montre comment la conception fait ce contrôle. Elle extrait la représentation de *BookType*, qui est passée au *PatronCategory* qui fait une comparaison. Donc, la représentation utilisée pour *BookType* est cruciale pour l'exactitude du code de tout le programme. Supposant que le *BookType* d'"enfant" devient "Enfant". Si on modifie au niveau de *BookType* sans apporter la même

modification dans *PatronCategory*, cela génère une erreur au niveau de ce dernier.
Ce qui nous emmène à dire que, pour qu'on puisse comprendre *BookType* il faut qu'on comprenne *PatronCategory*. Alors il y a une relation de couplage entre les deux classes. Et le couplage n'est pas de type STC, et cela parce qu'on remarque qu'il s'agit exactement du même modèle d'appel entre les classes E et C de l'exemple précédent.

Ce type de relation est similaire au concept de chaîne d'utilisation de définition (*use-def chains*), et ce type de couplage est nommé : « couplage indirect *use-def* ».

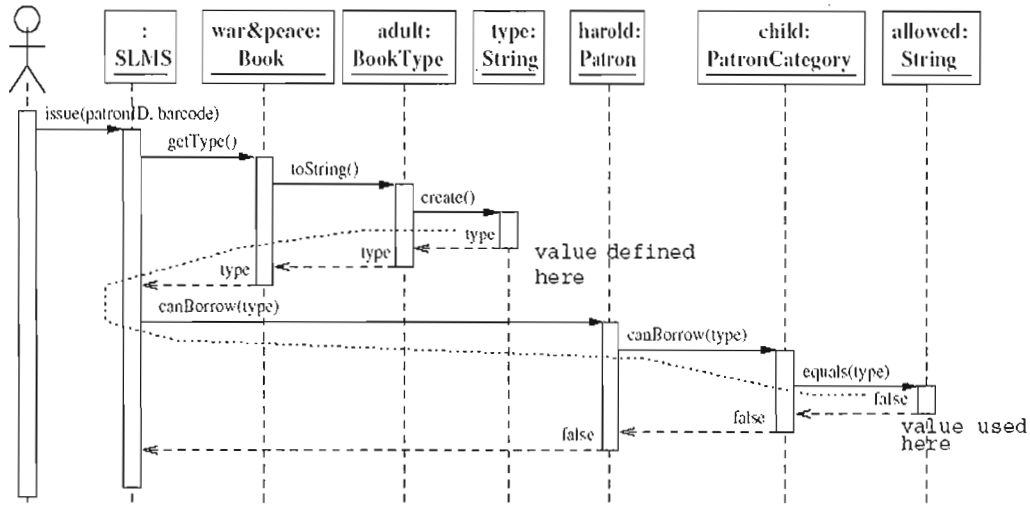


Figure A.60 Diagramme de séquence montrant une partie du comportement qui vérifie que le client est permis d'emprunter le livre indiqué

A.3.5.4 Détecteur de couplage indirect

L'outil de détection du couplage indirect (*Indirect Coupling Detector ICD*), est un plugiciel d'Éclipse. Cet outil détecte une forme particulière du « couplage indirect *use-def* ». En effet, cet outil implémente un algorithme, présenté dans la figure ci-dessous. Le but de cet algorithme après lui avoir passé, l'usage d'une variable v, est de trouver tous les endroits où a

été définie une variable donnée. Le point d'entrée de cet algorithme est la fonction *trackDef* (v, M), l'argument v correspond à l'usage d'une variable avec un nom spécifique et une position spécifique. L'autre argument M correspond à la méthode de déclaration contenant v.

L'analyse du code source pour le couplage indirect peut être faite à plusieurs niveaux, et peut s'étendre en allant d'une simple méthode au projet en entier. Une fois l'analyse terminée, *ICD* affiche les résultats sous forme d'une arborescence, dont le premier niveau représente l'usage de la variable et ses enfants correspondent aux modèles de flux de données.

La nature récursive de cet algorithme, cause à chaque fois une recréation de la même chaîne *use-def*, mais l'outil de détection évite cela.

Les auteurs ont appliqué *ICD* sur plusieurs projets existants d'à peu prés 25000 lignes de code, et 130 classes et interfaces. Les résultats étaient vraiment intéressants. Et les couplages indirect *use-def* ont bien été détectés par l'outil. Sauf, que lorsqu'une classe X a ce type de couplage avec une classe Y, et cette dernière couplée selon ce type de couplage avec une classe Z, et il y a une interaction entre les deux chaînes *use-def*, ce qui veut dire qu'il y a une forme de couplage indirect entre les classes X et Z, mais l'outil courant ne signale rien.

```
/**
   * v - variable usage
   * M - method declaration containing v
   */
trackDef(v, M) {
  E = find nearest previous definition of
      (i.e. assignment to) v within M
  if (E was found) {
    decide(E,M)
  } else {
    if (v is parameter to M)) {
      invocs = find statements/expressions
               that invoke M
      for (each inv in invocs) {
        argExpr = get the argument to inv
                  that corresponds to the
                  parameter v
        decide(argExpr, method declaration
                        containing inv)
      }
    } else if (v is a field) {
      fieldMods = find relevant expressions
                  that v gets defined as
                  (i.e. assigned to)
      for (each fieldMod in fieldMods) {
        decide(fieldMod, method declaration
                         containing fieldMod)
      }
    }
  }
}
/**
   * M - method declaration
   */
trackMethodResult(M) {
  E = find return value of M
  decide(E,M)
}

/**
   *E - expression
   *M - method declaration containing E
   */
decide(E,M) {
  if (E is a literal or constructor call) {
   Mark E as ultimate definition
  } else if (E is a variable) {
   trackDef(E,M)
  } else if (E is a method invocation) {
   trackMethodResult(declaration of the
                     method that E invokes)
  }
}
```

Figure A.61 Algorithme de détection de couplage indirect *use-def*

A.3.5.5 Conclusion et travaux futurs

Les travaux futurs proposés par les auteurs sont :

- Procéder à des études pour pouvoir déterminer comment le couplage indirect *use-def* est répandu, et le rapport qu'a ce type de couplage avec la maintenabilité.
- Concevoir une métrique qui dira s'il y a une corrélation entre le couplage indirect et la maintenabilité. Cette métrique est sous forme d'une échelle à intervalles.
- Approfondir la recherche et l'étude du couplage indirect pour découvrir d'autres formes qui impactent la conception du logiciel.

A.3.5.6 Références

Les auteurs présentent 16 références dont :

- Six (6) livres. Ces livres, à part celui de Yourdon et Constantine (1979) et celui de Fenton et Pfleeger (1997), n'ont que des relations indirectes avec le couplage.
- Neuf (9) articles de revues. Deux (2) seuls articles ont été publiés dans les années 2000, dont uniquement un (1) qui est directement lié au couplage. Et parmi l'ensemble de ces articles, trois (3) liés au tranchage.
- Un site web, il s'agit du site d'Éclipse : Eclipse Foundation, www.eclipse.org, 2004.

A.3.6 Kaung, Kham etThein (2005) « Pour visualiser le couplage entre les modules »

A.3.6.1 Introduction

Les auteurs voient le couplage comme un « indicateur » de certains « attributs de qualité » comme : la fiabilité, la maintenabilité, la facilité de compréhension, etc.

Ils le définissent comme « *une mesure de l'interdépendance entre deux modules* ». Nous savons que cette interdépendance peut avoir plusieurs aspects, mais les auteurs ne s'intéressent qu'à sa forme « donnée ». Autrement dit, les huit (8) métriques proposées vont mesurer les flux de données des modules (fonction ou procédure).

A.3.6.2 Direction de couplage

Le couplage d'un module donné peut avoir deux directions, soit importation ou exportation, cela est déterminé en fonction du sens des messages par lesquels il est concerné. S'il est censé recevoir ces messages, donc il s'agit d'un couplage d'importation, sinon, si c'est lui qui les envoie, donc c'est du couplage d'exportation.

À partir de la direction du message, et du type de donnée contenue dans ce message, huit métriques (ou directions) de couplage sont proposées :

9. « ***Couplage d'importation d'appel*** *- un module appelle un autre module mais il n'y a aucun paramètre, variables de référence communes*

10. ***Couplage d'importation scalaire*** *- une variable scalaire dans un autre module est passée comme paramètre effectif au module appelé A.*

11. ***Couplage d'importation de timbre*** *- un enregistrement dans un autre module est passé comme paramètre effectif au module appelé A.*

12. ***Couplage d'importation de tramp*** *- un paramètre formel dans un autre module est passé au module B comme paramètre effectif, le module B passe ensuite le paramètre formel au module A sans que B aie accédé ou modifié la variable.*

13. ***Couplage d'exportation d'appel*** *- un module appelle un autre module mais il n'y a aucun paramètre, variables de référence communes.*

14. ***Couplage d'exportation scalaire*** *- une variable scalaire dans le module A est passée comme paramètre effectif à un autre module.*

15. ***Couplage d'exportation de timbre*** *- un enregistrement dans le module A est passé comme paramètre effectif à un autre module.*

16. ***Couplage d'exportation de tramp*** *- un paramètre formel dans le module A est passé au module B comme paramètre effectif, le module B passe ensuite le paramètre formel correspondant à un autre module sans qu'il ait accédé ou modifié la variable.* »

Nous donnons un exemple de couplage d'importation et d'exportation de *tramp*, soient les algorithmes des modules Saisie, Affichage et Calcul, présentées dans le Tableau A.40 :

Tableau A.40 Algorithmes des modules Saisie, Affichage et Calcul

Module Saisie	Module Affichage	Module Calcul

Vide A ()
{ Entier x;
Saisir x au clavier;
Affichage(x);
}
	Vide B (entier y)
	{ Entier double;
	Double = Calcul(y);
	Afficher double à l'écran;
	}
		Entier C (entier z)
		{ Entier res;
		res = z*2;
		Retourner (res);
		}

Le module Saisie passe x comme paramètre au module Affichage. Ce dernier, sans le toucher, le passe à son tour au module Calcul, qui l'utilise dans un calcul d'une variable de sortie qu'il est censé retournée. Donc, il y a un couplage d'exportation *tramp* au niveau du module Affichage, et un couplage d'importation *tramp* au niveau du module Calcul.

A.3.6.3 Vue globale de l'architecture du système

Afin d'automatiser ces huit (8) métriques, les auteurs réalisent un système —dont la structure est présentée par la figure ci-dessous, qui est capable d'appliquer ces métriques sur n'importe quel programme C. Ledit Système comprend trois composants majeurs :

1. Analyseur syntaxique : ce composant fournit la structure d'un module donné, en extrayant les définitions et utilisations de toutes les variables, les points d'appel, les paramètres et leurs types. Toutes ces informations sont rassemblées dans un dépôt d'information propre au module analysé.

Analyseur de couplage : ce composant utilise l'information fournie par l'analyseur syntaxique pour déterminer la direction du couplage de deux modules. Ce composant implémente l'algorithme *FindDirectionCoupling*. Cet algorithme est présenté dans l'article. Voici, en bref le fonctionnement de cet algorithme :

- Dans un premier lieu cet algorithme va voir s'il s'agit d'un couplage *tramp* entre les deux modules A et B. C'est le cas lorsque A passe un paramètre à B qui ne l'utilise pas mais le passe à son tour à un autre module.

- S'ils ne sont pas de ce type de couplage, il va voir s'ils sont de couplage timbre ou scalaire et cela en vérifiant si lors de l'appel du module B par le module A, il y a eu un passage de paramètre.
    - Si ce paramètre est un enregistrement, alors le couplage est timbre.
    - Si c'est un simple scalaire, alors le couplage est de type scalaire.
    - Sinon, il n'y a aucun passage de paramètre, alors le couplage est dans ce cas est dit couplage d'appel.

2. Visualiseur (visionneur) de couplage : ce composant permet, par l'intermédiaire d'une interface utilisateur, de visualiser la direction du couplage d'un couple de modules. L'utilisateur doit d'abord choisir le couple de modules, ensuite ce composant va chercher la direction de couplage de ces deux modules dans le fichier fourni par l'analyseur de couplage. Après, il construit un graphe dirigé dans lequel il représente la structure du couplage de ce couple de modules et l'affiche à l'utilisateur.

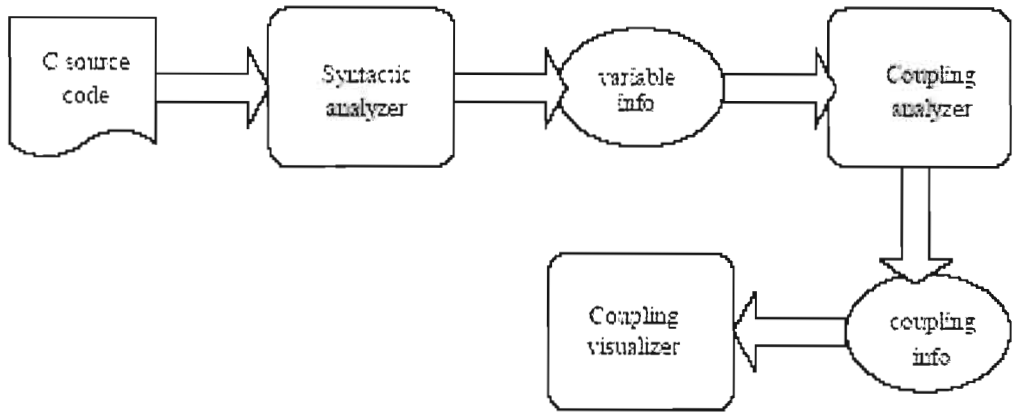


Figure A.62 Structure du système

A.3.6.4 Expérimentation

Par la suite, les auteurs testent leur système en le faisant fonctionner sur quelques programmes C. Chacun de ces programmes comprend un certain nombre de fichiers, de

procédures et de nombre de lignes. Pour chaque programme, ils examinent le type de couplage existant entre chaque paire de procédures (modules).

A.3.6.5 Conclusion

Finalement, dans la conclusion, ils rappellent les quatre (4) contributions apportées par leur travail :

1. Définir huit (8) directions de couplage, qu'ils présentent comme des métriques de couplage.
2. Donner une définition à chacune de ces métriques.
3. Produire un algorithme qui mesure les directions de couplage entre toutes les paires de modules dans un programme.
4. Permettre la visualisation du couplage de modules.

A.3.6.6 Références

Dans cet article, les auteurs ont utilisé treize (13) références:

- Le livre de Yourdon et Constantine (1979)
- Dix articles:
    - Trois (3) des années 2000
    - Cinq (5) des années 1990, dont un a été analysé en détails, il s'agit de l'article de Chidamber et Kemerer (1994)
    - Un (1) des années 1980
    - Un (1) des années 1970
- La page du site web de *sun* qui présente le compilateur java de *Sun* http://www.suntest.com/JavaCC
- Nous n'avons pas pu connaître la nature de la huitième référence de l'article.

A.3.7 Atole et Kale (2006) « L'évaluation des principes de couplage et cohésion de paquet pour la prédiction de la qualité de conception orientée objet »

A.3.7.1 Introduction

Dans l'introduction, les auteurs considèrent qu'une bonne conception fait appel à la « modularisation » à laquelle sont liés les deux facteurs de qualité[112] : cohésion et couplage.

Ils considèrent, que le couplage concerne toujours un couple de modules[113], du fait qu'ils le définissent comme « *le degré d'interdépendance d'une paire de modules* », et la cohésion d'un module « *est degré de nécessité de ses composants pour exécuter la tâche du module* ». Ils présentent quatre (4) critères de logiciel comme indicateurs d'une conception décomposable : *rigidité, fragilité, mobilité* et *viscosité*. Et ils affirment de façon indirecte que le non-dépassement des seuils tolérables de ces critères mène à un logiciel modulaire avec de bons niveaux de cohésion et de couplage.

A.3.7.2 Discussion et interprétation

A.3.7.2.1 Cohésion

Pour définir le concept cohésion, il fallait revenir à la définition originale, donnée par Yourdon et Contantine (1979). Cette dernière est jugée subjective, car le terme « élément de traitement » qui y était utilisé, n'a pas été formellement défini. Et puisqu'on est dans le cadre de l'approche OO, une définition plus appropriée a été suscitée. En effet, les auteurs font correspondre la cohésion d'une classe donnée au « *degré auquel les méthodes dans cette classe sont liées entre elles* ».

A.3.7.2.2 Couplage

---

[112] Voir le Chapitre II.

[113] Ce que les auteurs veulent dire par module « une collection ''d'éléments de traitement'' qui interagissent ensemble pour calculer une sortie », et c'est aussi synonyme de ''Classe''.

Toujours en OO et au niveau classe, on restreint les sources du couplage à deux types de relation qui peuvent surgir entre les classes :

1. Relation de contenance : une classe qui contient une autre.
2. Relation d'interaction : lorsque les méthodes d'une classe s'instancient à partir des paramètres d'autres classes.

A.3.7.2.3 Principes de cohésion de paquet

En général, pour contrôler ces indépendances (que ce soit entre les classes ou les éléments d'une même classe), il faut adopter des principes de gestion d'indépendance de modules. Trois principes de cohésion de paquet ont été choisis :

- ***Le principe de version équivalente (The Release Equivalent Principe : REP)*** : « *Le « granule » de la réutilisation est le granule de version. Un élément réutilisable, que ce soit un composant ou un faisceau de classes ; ne peut pas être réutilisé à moins qu'il soit contrôlé d'une certaine manière par un système de version* ».

- ***Le principe de fermeture commune (The Common Closure Principle : CCP)*** : « *Les classes qui changent ensemble appartiennent au même groupe. Quand nous groupons les classes qui changent ensemble dans les mêmes paquets, alors l'impact du paquet d'une version à une autre sera réduit au minimum* ».

- ***Le principe commun de réutilisation (The Common Reuse Principle : CRP)*** : « *Des classes qui ne sont pas employées ensemble ne devraient pas être groupées ensemble* ».

Il existe plusieurs métriques de cohésion de classe, mais puisque les auteurs s'intéressent à la mesure de la cohésion de paquets ils emploient la cohésion relationnelle (H) proposée par Martin, qui capture la cohésion des classes à l'intérieur du même paquet, et qui se nomme cohésion relationnelle (H), on la calcule comme suit:

$$H = (R+1)/N$$

Où :

- R : le nombre de relations entre les classes du paquet, et,
- N : le nombre de classes dans le paquet.

A.3.7.2.4 Principes de couplage de paquet

Pour le couplage, trois principes de couplage de paquet parmi ceux existants ont été sélectionnés :

- ***Le principe des dépendances acycliques (The Acyclic Dependencies Principle : ADP)*** : « *Les dépendances entre les paquets ne doivent pas former des cycles. Si un cycle est identifié entre les paquets, cela implique qu'un nouveau paquet casse le cycle* ».

- ***Le principe des dépendances stables (The Stable Dependencies Principle : SDP)*** : « *La stabilité est liée à la quantité de travail exigée pour apporter une modification. Une manière sûre de rendre un paquet d'un logiciel difficile à modifier est de faire de sorte que d'autres paquets dépendent de lui. Un paquet avec un bon nombre de dépendances entrantes est très stable parce qu'il exige beaucoup de travail pour apporter les changements nécessaires à tous les paquets dépendants* ».

- ***Le principe d'abstraction stable (The Stable Abstraction Principle : SAP)*** : « *Les paquets stables devraient être des paquets abstraits. Nous avons une structure de paquets tels que les paquets instables en haut, et les paquets stables en bas. Les paquets en haut, sont instables et flexibles, mais il est très difficile de modifier ceux d'en bas* ».

Les auteurs ont utilisé les métriques de stabilité et d'abstraction de Martin (1997) pour mesure de couplage. Comme métrique de stabilité, on a :

- ***Couplage afférent (Ca)*** : Le nombre de classes en dehors d'un paquet et qui dépendent des classes de ce paquet. Il correspond au nombre de dépendances entrantes[114].

- ***Couplage efférent (Ce)*** : Le nombre de classes en dehors d'un paquet et dont dépendent les classes de ce paquet. Il correspond au nombre de dépendances sortantes.

---

[114] La direction des dépendances est vue par rapport aux classes du paquet.

- ***Instabilité (I)*** **:** Cette métrique est calculée comme suit :

$$I = (Ce / (Ca+Ce))$$

Les valeurs de cette métrique appartiennent à l'intervalle [0..1]. Elle est égale à 0, dans le cas où, le paquet est abstrait (exemple : le paquet Y dans la Figure A.63), c.à.d. ne compte aucune dépendance sortante, on dit que ce type de paquet est stable. Et elle vaut 1, quand le paquet ne compte aucune dépendance entrante, donc un paquet concret (exemple : le paquet X dans la Figure A.64) et dans ce cas, le paquet est dit instable.

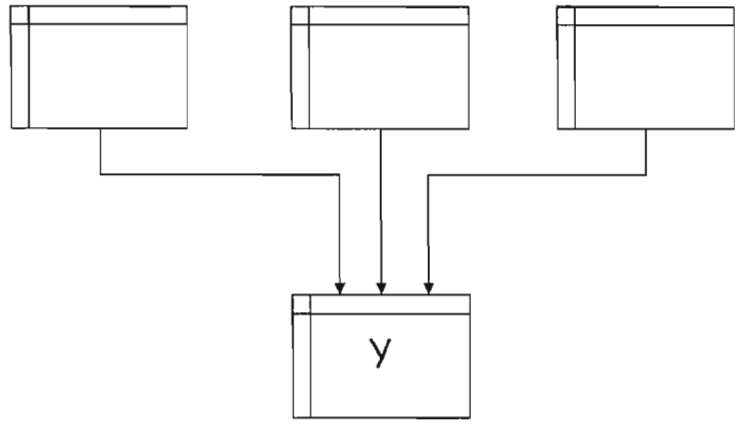


Figure A.63 Exemple de paquet abstrait

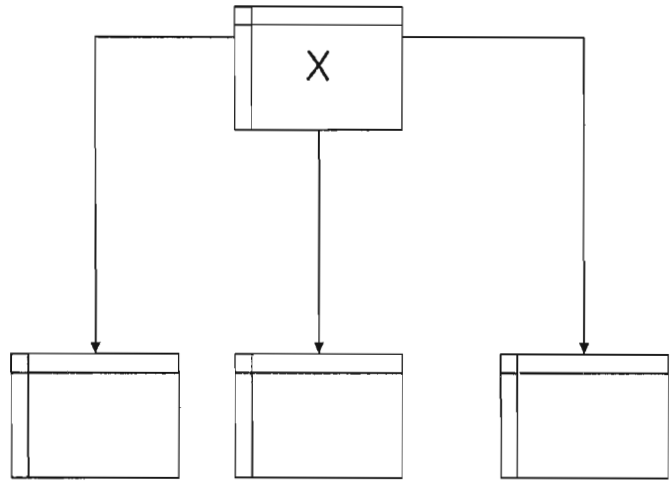


Figure A.64 Exemple de paquet concret

Et pour mesurer l'abstraction, on a :

- ***Abstraction (A)*** : A = (Na/Nc), avec Nc, le nombre de classes d'un paquet et, Na, le nombre de classes abstraites dans ce paquet..

Comme la métrique d'instabilité, les valeurs de cette métrique sont comprises dans l'intervalle [0..1]. Quand, le paquet ne comprend aucune classe abstraite, elle est égale à 0, et lorsque toutes les classes du paquet sont abstraites, elle vaut 1.

À partir des principes et des métriques de stabilité et d'abstraction, les auteurs arrivent à décrire une structure convenable d'une application OO, en effet, c'est une « *application composée de paquets concrets, instables, faciles à modifier, et de paquets abstraits, stables et faciles à étendre* ».

Le graphe A-I : On remarque une relation très évidente entre la métrique d'abstraction A et la métrique d'instabilité I. Lorsque A s'accroît, I se dégrade. On présente cette relation par le graphe A-I, donné par la Figure A.65.

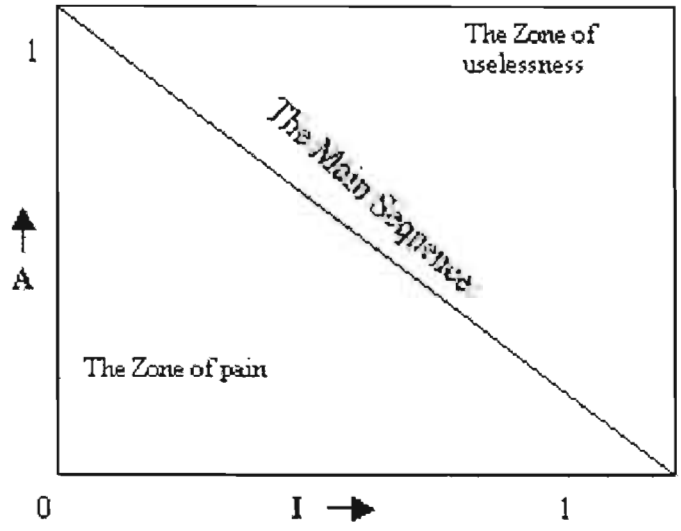


Figure A.65 Graphe A-I

Ce graphe est caractérisé par deux zones et une droite spécifiques à des cas particuliers :

9. ***Zone d'incompétence (Zone of uselessness) :*** C'est la partie en haut à droite du graphe, dans laquelle on trouve les paquets abstraits dont ne dépend aucun paquet.

10. ***Zone de souffrance (Zone of the pain) :*** cette zone est située en bas à gauche du graphe. Les paquets qui se retrouvent ici sont concrets et plusieurs autres paquets en dépendent.

11. ***Séquence principale (Main Sequence) :*** les paquets qui se localisent sur cette ligne, sont des paquets qui ont des dépendances sortantes et autres entrantes. Et reflètent une bonne conception.

Métrique de distance : La distance entre un paquet donné et la séquence principale est mesurable par une métrique, appelée métrique de distance (D).

$$D= A+I-1/\sqrt{2}$$

La métrique plus commode et normalisée de la distance (D'), est calculée comme suit:

$$D' = A+I-1$$

Quand cette métrique vaut 0, le paquet est dans la séquence principale, et quand il vaut 1, cela veut dire qu'il est très loin de la ligne de séquence principale.

A.3.7.3 Résultats et analyse

Dans leur étude de cas, les auteurs choisissent l'évaluation d'un exemple idéal de conception OO. Les résultats des métriques de cohésion, de couplage et de distance, et le graphe A-I du système, montrent que ses paquets ont un degré de cohésion élevé et un couplage faible.

A.3.7.4 Références

Dans cet article, les auteurs ont utilisé douze (12) références :

- Neuf (9) livres, dont celui de Yourdon et Constantine (1979)
- Trois (3) articles, dont deux qui datent des années 1990 et un des années 2000.

APPENDICE B

COHÉSION ET COUPLAGE : DÉFINITIONS ET CONSIDÉRATIONS D'AUTEURS REPUTÉS ET COMMENTAIRES PERSONNELS

Dans cet appendice nous avons choisi neuf (9) livres de GL publiés entre les années 1979 et 2003, desquels nous avons extrait les définitions et remarques sur le couple C&C. Dans la section B.1, nous présentons ces définitions. Par la suite, dans la section B.2, nous analysons ces définitions par l'entremise de comparaisons. Les neuf livres consultés sont :

- Les quatre éditions du livre « *Software Engineering A Practitioner s Approach* » de Pressman, publiées en 1982, 1987, 1992 et 1997.
- « *Software Engineering A Programming Approach* », écrit par Bell (2000).
- « *Fundamentals of Software Engineering* », écrit par Ghezzi, Jazayeri et Mandrioli (2003).
- « *Software Engineering An Object-Oriented Perspective* » de Braude (2001).
- « *Software Engineering Theory And Practice* » de Pfleeger, (2001).
- « *Software Engineering* » écrit par Jensen et Tonies (1979).

B.1 Définitions du couple C&C

Le tableau ci-dessous présente les définitions de « Cohésion » et de « Couplage » que nous avons retrouvées dans la littérature sélectionnée.

Tableau B.1 Définitions de « Cohésion » et de « Couplage »

Titre du livre	Nom (s) d'auteur(s) Année	Cohésion	Couplage
Software Engineering A Practitioner's Approach	**Roger S. Pressman, 1982**	Les notions d'éclatement et de condensation de modules sont abordées dans le cadre de l'amélioration du couplage et de la cohésion (mais ce n'est pas bien expliqué comme dans la version 1992) (p. 190) Dans la conception structurée, le mot. force fait référence à la cohésion de la conception structurée. (p. 233)	« *Les règles de couplage sont basées sur le fait que les données locales ne sont pas référencées. Le meilleur niveau de couplage, appelé le couplage de données, se produit quand toutes les données dont le module a besoin lui sont explicitement passées comme des paramètres.* » (p. 233)
Software Engineering A Practitione's Approach	**Roger S. Pressman, 1987**	La même définition et les mêmes remarques (p. 230 et p. 231)	La même définition et les mêmes remarques (p. 230 et p. 231)
Software Engineering A Practitione's Approach	**Roger S. Pressman, 1992**	« *Le concept de la cohésion peut être employé pour évaluer la simplicité d'une fonction donnée* » (p. 223)	« *Pour avoir un couplage, faible il faut que le nombre d'interfaces entre les modules soit petit (peu d'interfaces), et la quantité d'information qui circule via cette interface soit petite (petites interfaces). Quand les modules communiquent entre eux, cette communication doit être directe et claire (interfaces explicites)* » (p. 399)
Software Engineering A Practitioner's Approach	**Roger S. Pressman, 1997**	La même définition et les mêmes remarques. (p. 357 et p. 358) « *Puisque le logiciel conventionnel souligne la fonction comme mécanisme de localisation, les métriques se sont concentrées sur la structure ou la complexité interne des fonctions (par exemple, longueur de module, cohésion, ou complexité cyclomatique) ou la façon avec laquelle les fonctions se relient aux autres (par exemple, le couplage de module).*» (p. 665) « *Chaque méthode dans une classe, accède à un ou plusieurs attributs*	« *Le couplage entre les objets d'une classe (CBO) est le nombre de collaborations énumérés sur sa fiche de CRC.* » (p. 670) La métrique de couplage d'un module entoure le couplage de flux de données et de contrôle, le couplage global et le couplage de l'environnement. (p. 537)

Titre du livre	Nom (s) d'auteur(s) Année	Cohésion	Couplage
		(également appelés des variables d'instance), le manque de cohésion entre les méthodes (LCOM) est le nombre de méthodes qui accèdent à un ou plusieurs mêmes attributs. Si aucune méthode n'accède aux mêmes attributs, alors LCOM est égale à 0. » (p. 670) Il existe des métriques à cohésion fonctionnelle forte (SFC) et faible (WFC). Les valeurs de ces métriques sont comprises entre 0 et 1. (p. 536)	
Software Engineering A Programming Approach	**Douglas Bell, 2000**	« *La cohésion est une question d'union. Quelle est la meilleure façon de grouper des actions ? Quelle est la meilleure façon de grouper des données ? Dans le mariage des données et des actions, qui sont les dominantes et qui sont les subalternes, où est ce qu'elles sont associées de façon à ce qu'elles soient égales ?* » (p. 98) « *La cohésion décrit la nature des interactions dans un module de logiciel* » (p. 98)	« *Nous sommes habitués à l'idée qu'un module appelle d'autres modules, mais quelles sont les autres types d'interaction (couplage) qui peuvent exister entre les modules ? Quels sont les bons types de ces interactions et quels sont les mauvais ?* » (p. 95)
Fundamentals of Software Engineering	**Carlo Ghezzi, Mehdi Jazayeri et Dino Mandrioli, 2003**	« *La cohésion est une propriété interne du module.* » (p. 48)	« *Le couplage caractérise la relation d'un module avec les autres modules. Il mesure l'interdépendance de deux modules.* » (p. 48)
Software Engineering An Object-Oriented Perspective	**Eric J. Braude, 2001**	« *La cohésion dans un module est le degré de la communication entre les éléments de ce module.* » (p. 251) « *Le faible couplage et la forte cohésion sont particulièrement importants pour la conception de logiciel en raison de la*	« *Le couplage décrit le degré avec lequel les modules communiquent avec d'autres modules.* » (p. 251)

Titre du livre	Nom (s) d'auteur(s) Année	Cohésion	Couplage
		nécessité de porter des modifications continuellement aux applications. » (p. 252)	
Software Engineering Theory and Practice	**Shari Lawrence Pfleeger, 2001**	La cohésion et le couplage sont des attributs indicatifs de la qualité de la conception, car ils influencent la facilité de modification qui est une caractéristique de la qualité du logiciel. (p. 220) « *La mesure du manque de cohésion entre les méthodes d'une classe, est le nombre d'intersections nulles de ces méthodes moins le nombre d'intersections non nulles.* » (p. 299) « *Les caractéristiques de la conception, telles que le faible couplage, la cohésion élevée, et les interfaces bien définies, devraient également être des caractéristiques de code, de sorte que les algorithmes, les fonctions, les interfaces, et les structures de données puissent être traçables dans les deux sens.* » (p. 310) Mieux montrer le couplage et la cohésion par un choix judicieux des noms des paramètres et par des commentaires. (p. 312)	Mêmes citations tirées des pages : 220, 310 et 312.
Software Engineering	**Randall W. Jensen et Charles C. Tonies, 1979**	« *La cohésion est la mesure du type de relations qui existent entre les éléments du même module.* » (p. 176)	« *On mesure le couplage non seulement en prenant en considération le nombre de connexions, mais aussi le type de connexion et le type d'information communiquée via cette connexion.* » (p. 175)

B.2 Discussion

B.2.1 Cohésion

Pressman (1992) a défini la cohésion comme « *une extension naturelle du concept de masquage de l'information* ». D'après Pressman, la cohésion n'est qu'une extension d'un principe de programmation modulaire qui mène au masquage des données entre les modules d'un logiciel. Donc, on ne parle plus d'un module mais de la relation de ce module avec les autres modules, puisque c'est l'accès aux données d'un module par les autres qui nous intéresse.

« *Nous sommes habitués à l'idée qu'un module fait appel à d'autres modules, mais quelles sont les autres types d'interaction (couplage) qui peuvent exister entre les modules ? Quels sont les bons types de ces interactions et quels mauvais ?* » (p. 95)

Un peu plus loin, il définit un module cohésif comme « *un module qui n'exécute qu'une seule et simple tâche ayant peu d'interactions avec les autres parties du logiciel* ». Donc, on insiste toujours sur les interactions qui surgissent entre les modules et qui, de préférence, ne doivent pas être complexes et nombreuses, ainsi que sur la fonction remplies par le module qui doit être unique et simple. Donc, au début, Pressman s'est contenté de restreindre la cohésion à une description d'interactions des modules, puis il s'est rattrapé en disant que c'est aussi une description de la force fonctionnelle d'un module. C'est comme si la cohésion est une caractéristique d'un module et en même temps une caractéristique des relations entre les modules, ce que nous trouvons un peu étrange.

Pressman a donné deux définitions de module, la première dit qu'« *un module est une unité modulaire linguistique qui correspond à une unité en langue parlée* », la deuxième dit, *qu'« un module est une manifestation générique d'un sous-programme qui peut être selon le langage de programmation une procédure, une fonction, etc.* ».

Tandis que Bell (2000) a défini le concept de cohésion en relatant sa fonction, qui est de « *décrire la nature des interactions dans un module* ». Ainsi, la cohésion décrit le type de relations entre les composants ou les éléments d'un module. Contrairement à Pressman, Bell

ne fait en aucun lieu référence aux interactions entre les modules. D'autre part, il nie que la cohésion soit une métrique ou une mesure. Donc la cohésion est loin d'être un concept mesurable, mais c'est plutôt une sorte d'évaluation qualitative de la qualité du logiciel.

Voyons comment il définit le terme module. « *Un module est un morceau assez indépendant d'un programme qui a un nom, quelques instructions, et quelques données qui lui sont propres. Module peut correspondre à des tas de choses et cela en fonction du langage de programmation* ». Cela peut être, un sous-programme, une procédure, une fonction, une méthode, un paquet, une classe, un objet, etc.

Alors que Braude (2001) présente la cohésion comme « *le degré de communication qui s'effectue entre les éléments du module* ». Autrement dit, Braude présente la cohésion comme un degré, ordre ou état de la communication entre les éléments d'un module. Donc elle détermine la force de la liaison des composants d'un module.

Braude (2001) et Bell (2000) sont d'accord sur le fait que la cohésion décrit en quelque sorte les relations entre les éléments d'un module. Mais, Bell affirme que cette description concerne la nature des interactions, tandis que Braude souligne la description du degré des interactions. Bell parle de nature car il insiste sur le fait que la cohésion n'est pas une mesure mais plutôt une évaluation qualitative. Alors que Braude parle de degré parce qu'il voit (c'est dit implicitement via le mot degré utilisé dans la définition) que la cohésion est une mesure définissable à partir du type ou de la nature des relations qui existent entre les éléments du même module. Un peu plus loin, Braude souligne l'importance de la cohésion dans la conception, puisqu'elle détermine la facilité de modification du logiciel.

Pfleeger (2001) définit la cohésion comme un attribut de la qualité de la conception puisque sa force et sa faiblesse influencent la facilité de modification. Donc, il a défini la cohésion en spécifiant son impact sur la qualité du logiciel. Alors on doit être capable de tracer les fonctions et les interfaces en prenant en considération ce concept durant toutes les phases du CVL. Et, il définit un module (ou composant) comme « *une entité avec des*

entrées, des sorties, ou des caractéristiques bien définies ». Il définit aussi un module bien défini comme « *un composant dont chaque sortie est un résultat du fonctionnement du module et pour lequel aucune entrée ne devient un résultat sans avoir été transformée d'une manière quelconque par le module* ».

Quant à Jensen et Tonies (1979), ils définissent la cohésion exactement comme l'a définie Braude (2001) auparavant, sauf que leur définition est plus précise que celle de Braude, car ils utilisent à la place de « degré » une mesure du type et à la place de « communication » les relations. Donc la cohésion est définissable numériquement lorsque le type de relations entre les éléments du module est déterminé.

Ghezzi, Jazayeri et Mandrioli (2003) définissent la cohésion comme une propriété interne au module et considèrent qu'« *un module est fortement cohésif lorsqu'il y a une forte liaison entre ses éléments et que ces derniers coopèrent tous pour réaliser un but, qui est la fonction du module* ». Dans cette définition, ils retracent la force fonctionnelle du module déjà mentionnée par Pressman (1992) et aussi la relation entre les éléments du module mentionnée par tous sauf Pressman (1992).

B.2.2 Couplage

Pressman (1992) voit le couplage comme une mesure de la dépendance fonctionnelle entre les modules d'un logiciel, et, cette dépendance ou interdépendance est évaluée en fonction de :

- La complexité des interfaces qui lient les modules entre eux (petit nombre d'interfaces, petites interfaces et interfaces explicites),
- Les points d'entrée aux modules, et
- La nature des données échangées (des données pures, des contrôles ou des contrôles masqués).

En ce qui concerne Bell (2000), il définit le couplage comme un moyen de classification de la description des interactions entre les modules. Le degré de couplage est

déterminé à partir du type ou de la nature des interactions entre les modules et cette nature dépend de la façon avec laquelle les modules interagissent. Il est de même affecté par la taille de l'interaction, c'est-à-dire le nombre d'objets qui connectent les modules.

Bell (2000) a préféré relater explicitement les différentes formes de communication entre les modules (comme « *changement du code d'un autre module, données globales ou partagées, procédure d'appel avec des données de paramètres pures, passage d'un flux de données séquentiel* », etc.) et qui déterminent indirectement le couplage du logiciel, au lieu de lister les critères influençant le couplage, comme l'a fait Pressman (1992).

Braude (2001) trouve qu'un couplage faible est d'une importance primordiale dans la conception du logiciel et cela à cause « *des modifications continuelles à apporter* ». Il considère le couplage comme une « *description du degré de communication inter-modulaire* ».

Pfleeger (2001) évoque la question de la facilité de communication comme Braude (2001), mais non pas pour souligner le rôle d'un couplage fort pour la conception, mais plutôt pour préciser ce que représente ce concept, qui n'est autre qu'un attribut indicatif de la qualité de la conception.

Pour Jensen et Tonies (1979), le couplage est « *une mesure des relations qui existent entre les modules* », et cette mesure se détermine à partir du :

- Nombre de connexion,
- Type de connexion, et
- Type d'information échangée lors de cette connexion.

Si on suppose que le terme « connexion » est synonyme d'interface, on remarque que Pressman (1992) a cité les mêmes facteurs influençant le degré de couplage d'un logiciel que ceux cités ci-dessus. Sauf qu'il a ajouté un autre facteur qui est le point d'entrée au module que Jensen et Tonies (1979) ont négligé.

Ghezzi, Jazayeri et Mandrioli (2003) prennent le couplage pour une caractéristique de la relation qui lie un module aux autres modules du logiciel et en même temps le considèrent comme une mesure de l'interdépendance de deux modules. Donc, le couplage mesure les relations et l'interdépendance modulaire.

APPENDICE C

RÉSUMÉS D'ARTICLES LIÉS

Dans cet appendice, nous résumons quatre (4) articles qui concernent la complexité et qui sont utiles pour une meilleure compréhension des études de C&C.

Le premier est celui de Weyuker (1988) dans lequel l'auteur présente les célèbres propriétés que doit vérifier une métrique de complexité logicielle et qui ont, par la suite, été très utilisées dans la validation de ce genre de métriques. Le deuxième, celui de Basili et Hutchens (1980), présente une famille de métriques de la complexité structurelle. Nous avons choisi d'étudier cet article afin d'avoir une idée sur les recherches qui ont abordé la complexité structurelle, sans pour autant considérer le couple C&C. Enfin les deux derniers sont écrits par Kushwaha et Misra (2006) et traitent de la complexité cognitive abordée dans le chapitre 3. Dans ce qui suit, pour chacun de ces quatre (4) articles, nous présentons le but et un résumé du contenu.

C.1 Weyuker (1988) « Évaluation d'une métrique de complexité logicielle »

C.1.1 But

Le but de cet article est de proposer un cadre d'évaluation des métriques de complexité logicielle.

C.1.2 Résumé

L'auteur commence par une introduction dans laquelle elle aborde les points suivants :

- L'apparition d'une multitude de mesures de la complexité du logiciel de nature syntaxique.
- Les études informelles de ces mesures qui ne font pas vraiment avancer en quoi que ce soit ces mesures.
- L'ignorance complète ou partielle de ce que ces mesures sont supposées mesurer constitue l'une des difficultés d'évaluation de ces mesures.

Ce qui manque, c'est une étude formelle de l'évaluation de ces métriques. L'article présente un ensemble de propriétés qu'une métrique de complexité doit avoir pour pouvoir dire qu'elle mesure ce qu'elle est censée mesurer. Ces propriétés permettent au programmeur d'affiner le choix de la mesure adéquate à adopter, et aussi d'évaluer les nouvelles mesures qui seront conçues et cela sans se retourner vers des études empiriques. Donc, Weyuker (1988) pense qu'on peut se restreindre au cadre d'évaluation qu'elle présente pour jauger l'efficacité d'une métrique de complexité.

Avant de présenter les propriétés, on présente les mesures dont la comparaison d'utilité s'avère intéressante, et qui seront évaluées à chaque fois qu'on cite une propriété. Ces mesures sont :

- ***Le nombre de lignes du code :*** mesure très ancienne connue pour sa simplicité.
- ***Le nombre cyclomatique de McCabe (1976)*** : qui se calcule comme suit :

$$v = e - n + 2p$$

Avec : e, le nombre d'arêtes, n, le nombre de sommet et p, le nombre de composants connectés.

- ***La mesure de l'effort :*** une mesure établie par Halstead (1977) :

$$E = (n1*N2)*(N1 + N2)*\log(n1+n2) / (2*n2)$$

Avec : n1, le nombre d'opérateurs distincts, n2, le nombre d'opérandes distincts, N1, le nombre total d'opérateurs et N2, le nombre total d'opérandes.

- ***Knot measure*** (Woodward, Hennell et Hedley, 1979) *:* la complexité du flux de contrôle du programme en entier est égale la somme des complexités des flux de

contrôle des blocs constituants ce programme. La complexité du flux de contrôle de chaque bloc est :

$$Dfi = \sum DEF(vj)$$

Avec : j allant de 1 à || Vi||), DEF (vj), le nombre de définitions disponibles de la variable vj de l'ensemble Ri, qui représente l'ensemble de définitions de variables atteignables par le bloc ni et || Vi ||, la cardinalité de l'ensemble Vi.

La complexité du flux de données du programme en entier est :

$DF = \sum DFi$ (avec: i allant de 1 à || S ||)

Avec : || S ||, l'ensemble des blocs du programme.

Dans ce qui suit, nous présentons les neuf (9) propriétés proposées par Weyuker (1988), avec l'évaluation des quatre métriques de complexité vis-à-vis de chacune des neuf propriétés :

Propriété 1 : $(\exists P)\ (\exists Q)\ (|P| \neq |Q|)$.

La première propriété, est une condition assez transparente à imposer pour affirmer en quelque sorte l'utilité de l'existence de la mesure. En fait, « *une mesure qui évalue tous les programmes comme égaux n'est pas vraiment une mesure* ».

Cette propriété caractérise les quatre (4) mesures présentées précédemment, chose qui n'est pas très étonnante, car ça reste une condition nécessaire mais pas suffisante pour approuver une mesure.

Propriété 2 : Soit c un nombre non négatif. Il existe seulement un nombre fini de programmes de complexité c.

Cette propriété, concerne la sensibilité de la mesure. « *Une mesure n'est pas suffisamment sensible si elle divise tous les programmes en très peu de catégorie* ».

La mesure du nombre de lignes de code satisfait cette propriété si et seulement si on suppose qu'« *il y a un certain très grand nombre possible qui peut être représenté et une*

limite supérieure de la longueur d'une instruction, et que cette limite peut être une fonction de la machine utilisée, et elle est supposée exister ».

Le nombre cyclomatique ne satisfait en aucun cas cette propriété, car elle ne fait pas de différence entre les programmes qui font des calculs nombreux et complexes et ceux qui en font moins. Ceci reflète « évidemment et intuitivement » sa faiblesse.

En ce qui concerne la mesure de l'effort, elle satisfait cette propriété, mais uniquement dans le cas ou $N2(P) \geq n2(P)$.

Pour la complexité du flux de contrôle, cette propriété n'est pas satisfaite.

Propriété 3 : Il y a des différents P et Q tels que : $|P|^{115} = |Q|$

La troisième propriété vient préciser la deuxième, car il est vrai qu'on ne veut pas que la mesure divise les programmes en un petit nombre de catégorie, mais on ne veut pas non plus que deux programmes différents ne puissent pas avoir la même valeur de la mesure.

Il est clair que les quatre (4) mesures de complexité satisfont cette propriété.

Propriété 4 : $(\exists P)\ (\exists Q)\ (P \equiv Q\ \ \&\ |P| \neq |Q|)$

La quatrième propriété, part de l'idée que la complexité d'un programme ne dépend pas vraiment de la fonction remplie par ce dernier mais plutôt des « *détails de mise en œuvre de cette fonction* ».

Étant donné que les quatre (4) mesures dépendent de la mise en œuvre du programme, alors elles satisfont toutes cette propriété.

Propriété 5 : $(\forall P)\ (\forall Q)\ (|P| \leq |P\ ;\ Q|\ \&\ |Q| \leq |P\ ;\ Q|)$

---

[115] $|P|$ désigne la valeur de la métrique pour le programme P.

La cinquième propriété met en évidence le rapport « complexité non-absolue » du programme et de ses composants, tel que, les composants du programme ne peuvent pas être plus complexes que le programme lui-même.

Il est tout à fait normal de considérer cette propriété comme une particularité de la mesure du nombre de lignes de code et du nombre cyclomatique, puisque :

- Dans le cas du nombre de lignes de code : $(\forall P)\ (\forall Q)\ (|P ; Q| = |P| + |Q|)$
- Dans le cas du nombre cyclomatique : $(\forall P)\ (\forall Q)\ (|P ; Q| = |P| + |Q| - 1)$

En ce qui concerne la mesure de flux de contrôle, elle ne satisfait pas cette propriété et cela est dû « *au flux de données de l'inter-bloc qui amplifie encore plus la complexité* ».

La non satisfaction de cette propriété par la mesure de l'effort remet en question l'utilité de cette dernière, car il s'avère aberrant que l'effort fourni pour écrire le programme en entier est inférieur à l'effort fourni pour écrire une partie de ce programme.

Propriétés 6 : a : $(\exists P)\ (\exists Q)\ (\exists R)\ (|P| = |Q|\ \&\ |P ;R| \neq |Q ;R|)$ et b : $(\exists P)\ (\exists Q)\ (\exists R)\ (|P| = |Q|\ \&\ |R ;P| \neq |R ;Q|)$

Si un programme R peut être concaténé à deux programmes P et Q ayant la même complexité a) les programmes résultants (P;R) et (Q;R) et b) (R;P) et (R;Q) n'ont pas nécessairement la même complexité.

Le nombre de lignes de code et le nombre cyclomatique ne satisfont pas cette propriété, tandis que la complexité du flux de contrôle et la mesure de l'effort la satisfont.

Propriété 7 : Il existe deux programmes P et Q constitués des mêmes instructions placées dans un ordre différent, de telle sorte que : $(\exists P)\ (\exists Q)\ (|P| \neq |Q|)$.

La septième propriété affirme l'influence de la complexité du programme par l'ordre de ses instructions.

Nombre de lignes de code, nombre cyclomatique et l'effort ne satisfont pas cette propriété.

Propriété 8 : Si P est un deuxième nom du programme Q alors, $|P| = |Q|$.

La huitième propriété vient souligner que le nom du programme n'influence pas sa complexité.

Les quatre (4) mesures considérées satisfont cette propriété.

Propriété 9 : $(\exists P)\ (\exists Q)\ (|P|+ |Q| < |P\ ;\ Q|)$.

La neuvième propriété affirme que la complexité d'un ensemble de sous-programmes interagissant ensemble, est toujours plus grande que la somme des complexités de ces mêmes sous-programmes mis à part.

Les mesures de lignes de code et le nombre cyclomatique ne satisfont pas cette propriété, tandis que les mesures de flux de contrôle et de l'effort la satisfont.

C.2 Basili et Hutchens (1980) « Étude d'une famille de métriques de la complexité structurelle »

C.2.1 But

Cet article présente une formule définissant une métrique de complexité qui, en fixant les valeurs de certains de ses paramètres, on retrouve les complexités « classiques ».

C.2.2 Résumé

Les auteurs commencent leur article en considérant les trois attributs de complexité du logiciel les plus répandus :

1. Le volume.
2. L'organisation des contrôles.

3. L'organisation des données.

Il existe plusieurs métriques propres à chacun de ces attributs, mais malheureusement bien des concepteurs n'ont pas été convaincus par l'utilisation de ces dernières dans le processus d'assurance qualité pour les deux raisons suivantes :

- On ne sait pas à quelle phase du CVL elles peuvent être appliquées.
- On ne peut pas les contextualisées ; c'est-à-dire, on ignore exactement les caractéristiques du logiciel[116] dont elles dépendent.

Donc, il s'est avéré important de définir une famille de métriques paramétrisées qui prend en considération certains éléments des attributs définissant la complexité. Cette famille, qui n'inclut pour le moment que les concepts de volume et de contrôle, est présentée comme suit :

$$C(p) = b \sum c(pi) + f(n, l, t, s) \text{ (avec: i allant de 1 à k)}$$

Avec :

- p : le programme (qui peut être lui-même un sous-programme) qui est décomposé en k sous-programmes (pi).
- b : génère le multiplicateur pour chaque niveau d'imbrication.
- f : la clé de la mesure.
- n : le nombre de décisions contenues par le programme.
- l : le niveau d'imbrication du programme.
- t : le type de structure instanciée par le programme (comme les instructions : *while*, *case* et *if*).

---

[116] Ici ce que nous voulons dire par caractéristiques du logiciel, c'est les variables qui définissent la typologie du logiciel, comme l'envergure du projet, le domaine de fonctionnement, le type des utilisateurs, etc.

- s : la qualité structurelle du programme (ce paramètre répond à la question : est-ce qu'il s'agit d'un programme structuré ?). Il permet de subdiviser les programmes en deux grandes catégories : programmes structurés au sens strict (s=1) et programmes simplement structurés (s=0). Les auteurs font l'hypothèse que les programmes structurés au sens strict sont moins complexes[117].

À partir de cette famille de métriques, on peut générer quelques métriques déjà existantes, et cela en fonction des valeurs de paramètres de cette dernière. Le tableau ci-dessous résume ces différents cas :

Tableau C.1 Extraction des métriques existantes à partir de la nouvelle famille de métriques

Conditions	Nouvelle formule	Métrique ressortie
b = 1 Le programme p est convenablement structuré, et f (n, l, t) = g (n, l, t)[118] = n	C(p) = ∑ c (pi) + n	La Complexité cyclomatique moins un,
La décomposition du programme p est basée sur la structure syntaxique du langage.	C(p) = ∑ c (pi) + {1 si p est une instruction, 0 sinon} C(p) = ∑ c (pi) + {1 ; p est une expression ou un terme, 0 sinon}	La Mesure de Halstead (mesure de la taille du logiciel).
« *En combinant la distinction de programme structuré au sens strict et de programme simplement structuré avec les décompositions syntaxiques au niveau de l'instruction* »	C(p) = ∑ c (pi) + {0 si p est convenablement structuré, n sinon}	« *La complexité essentielle*[119] *avec les prédicats complexes traités comme de simples décisions* ».

[117] À ce propos, l'échange entre Dijstra et Knuth à propos des GOTO montre que l'hypothèse n'est pas nécessairement partagée par tous les informaticiens.

[118] On distingue deux types de programmes, des programmes convenablement structurés pour eux ladite fonction f(n, l, t, s) est égale à f(n, l, t), et d'autres non convenablement structurés qui pour eux f devient g(n, l, t)

[119] Les auteurs définissent la complexité essentielle comme : « *la complexité cyclomatique moins le nombre de sous-programmes obtenus de la décomposition principale (tout en ignorant les séquences).* »

C.3 Kushwaha et Misra (2006) « Les métriques de la complexité cognitive et leur impact sur la fiabilité du logiciel en se fondant sur des modèles cognitifs de développement de logiciel »

C.3.1 But

En 2006, Kushwaha et Misra publient cet article avec deux objectifs :

- Souligner l'importance des métriques cognitives et leurs impacts sur la fiabilité du logiciel.
- Déterminer les secteurs (les activités qui rentrent dans le processus de développement du logiciel) qui sont les plus importants dans la vie d'un logiciel fiable.

C.3.2 Résumé

La complexité est l'un des attributs qui influencent le plus la qualité de conception, car elle a un grand impact sur deux caractéristiques de qualité très importantes : la maintenabilité et la fiabilité.

La complexité du logiciel est mesurable dans n'importe quelle « phase » de développement et cela en termes de la compréhensibilité de ce logiciel. Cette dernière ne dépend pas seulement de quelques propriétés du logiciel, mais aussi de certains aspects cognitifs. Le modèle cognitif de développement proposé par Kushwaha et Misra (2006), présenté dans la Figure C.1, s'appuie sur ces aspects cognitifs. Il présente les différentes phases par lesquelles passe le logiciel afin de garantir une fiabilité plus ou moins élevée. Ces phases sont :

- Les phénomènes cognitifs basés sur le choix des membres d'équipe ;
- Génie d'exigences orienté personne ;
- L'inspection cognitive du logiciel (ça fait partie des tests) ;
- La documentation basée sur les phénomènes cognitifs et
- La validation par des métriques.

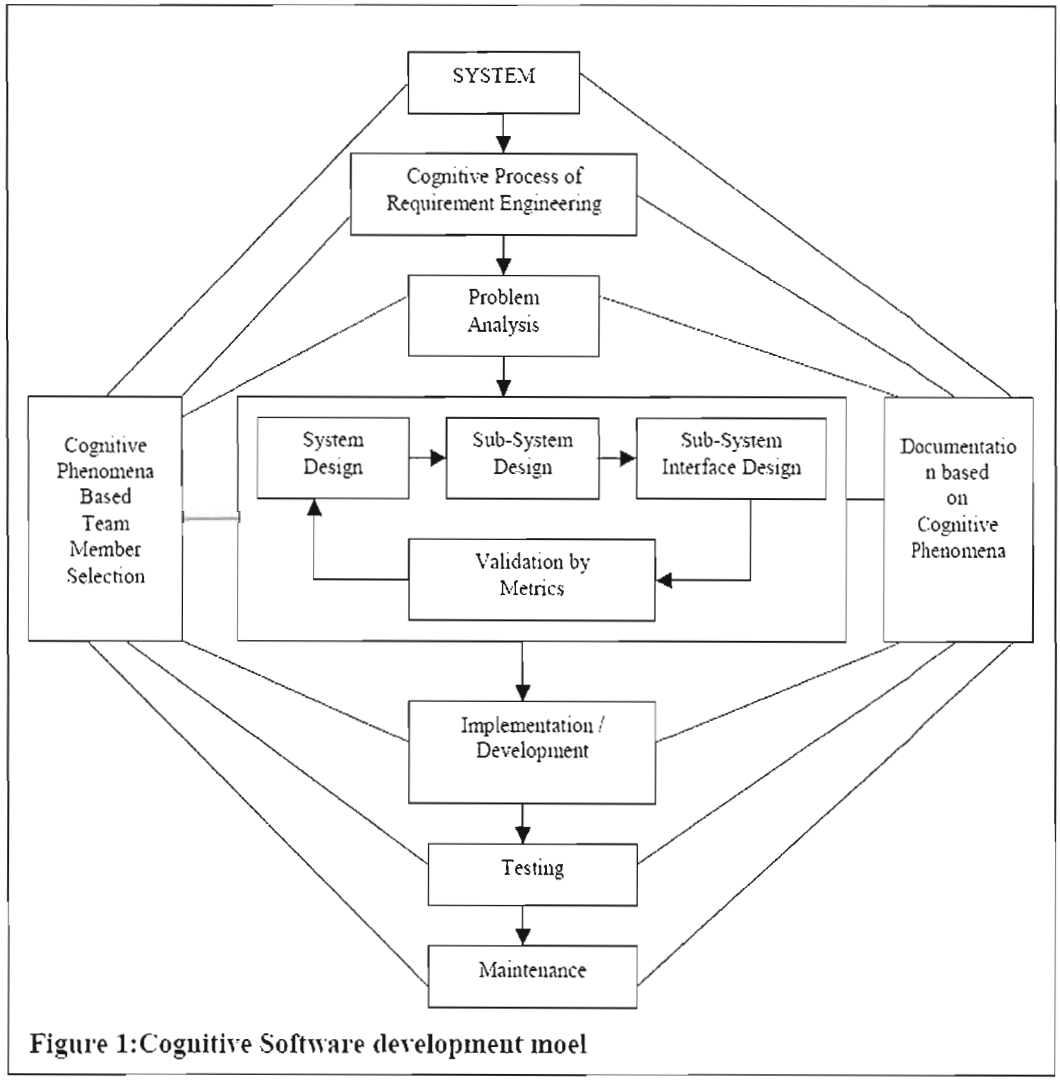


Figure C.1 Modèle cognitif du développement du logiciel

Lors du « *processus cognitif du génie d'exigences* », les analystes ignorent le rôle que joue l'utilisateur final dans l'élicitation des exigences et la détermination de certains traits du produit final, représenté par le logiciel et la façon avec laquelle il interagit avec l'utilisateur afin de rendre le service pour lequel il été créé. Pire encore, ces analystes vont jusqu'à attribuer le rôle de l'utilisateur final au client, ce qui peut certainement les induire en erreur, car le client se trouve souvent incapable de décrire le niveau cognitif et psychologique de

l'utilisateur final du logiciel. Et, même quand il se prétend capable de le faire, il le fait en se basant sur le système à mettre en œuvre et en négligeant l'appartenance de l'utilisateur final à ce dernier.

De ce fait, on propose un modèle cognitif du GL qui permet de fournir une description des caractéristiques cognitives intégrant celles des utilisateurs et via lesquelles les analystes vont regrouper les exigences collectées. Ce qui va leur permettre d'avoir une vision globale du système.

Le succès du processus « des inspections cognitives » découle de la diversité dans les niveaux cognitifs des membres de l'équipe d'inspection.

La complexité cognitive de la documentation du logiciel —ou autrement dit la compréhensibilité des artefacts produits, influence indirectement la fiabilité et la qualité du logiciel. Car la complexité cognitive a un impact direct sur la complexité globale du logiciel « *et sa mesure ne tient pas compte du respect des standards d'écriture imposés* ». Mais, elle tient plutôt compte de la quantité d'informations utiles pour qu'un utilisateur final, dont les caractéristiques cognitives sont toutes à un niveau très bas, puisse comprendre ce qui lui est présenté dans la documentation du logiciel.

On propose une métrique de la complexité cognitive, nommée complexité conceptuelle cognitive d'une classe/module (*Cognitive Conceptual Complexity of Class / Module* CCCC), et qui est calculée en fonction du nombre de concepts principaux dans un module et de leurs poids. Ce poids est déterminable selon son degré cognitif. La formule mathématique de cette métrique est la suivante :

$$CCCC = \sum_{i=1}^{m} [(\text{No. of distinct Cognitive Concepts}) * (\text{Weight of Concept})]_i$$

Avec m, le nombre de concepts distincts dans le module.

C.4 Kushwaha et Misra (2006) « Une mesure de la complexité cognitive de l'information du logiciel modifiée »

C.4.1 But

Le but de cet article est de développer une métrique « robuste » de la mesure de la complexité cognitive de l'information, nommée CICM, qui englobe tous les paramètres qui affectent le plus la difficulté de compréhension.

C.4.2 Résumé

L'article commence par citer la définition de complexité de la norme IEEE Std 610.12 (1990) : « *degré de difficulté de la compréhension et de la vérification de la conception ou de la mise en œuvre d'un composant ou du système* ». Après avoir souligné que plusieurs métriques de complexité ont été proposées et comme introduction à leur métrique, les auteurs se posent deux questions :

1. « *Qu'est ce qui rend un logiciel difficile à comprendre ?* »
2. « *Y'a-t-il une relation entre la difficulté de compréhension du logiciel et sa complexité cognitive ?* »

Avant de présenter leur métrique qui mesure la complexité cognitive de l'information (*Cognitive Information Complexity Measure* CICM), les auteurs présentent deux mesures de complexité existantes assez importantes.

La première mesure est la métrique de la complexité KLCID conçue par Klemola et Rilling (2003). Elle est calculée à partir du nombre d'identificateurs contenus dans le code et le nombre de lignes de code uniques, définies comme « *les lignes qui ont le même type et genre d'opérandes avec le même arrangement des opérateurs* ».

KLCID = nombre de lignes de code unique / nombre d'identificateurs dans l'ensemble des lignes uniques du code

Les auteurs jugent que la comparaison de chaque ligne de code aux autres lignes est une tâche qui demande beaucoup de temps et d'énergie. C'est ce qui rend « *cette mesure très difficile et longue à réaliser* ».

La deuxième mesure est la taille fonctionnelle cognitive (*Cognitive Functional Size* CFS), définie par Wang et Shao (2003). Elle est calculée à partir du nombre d'entrées, du nombre de sorties et du poids cognitif du programme. Ce dernier est calculé en fonction des structures de contrôle de base (*Basic Control Structures* BCS). Wang et Shao (2003) ont attribué pour chacune d'entre elles (les structures de contrôle de base) un certain poids cognitif.[120] Le tableau ci-dessous présente ces poids.

Tableau C.2 Poids cognitif de chaque structure de contrôle de base

Category	BCS	Weight
Sequence	Sequence (SEQ)	1
Branch	If-Then-Else (ITE)	2
	Case	3
Iteration	For-do	3
	Repeat-until	3
	While-do	3

CFS = (Ni + No) * Wc

Avec :

Ni : nombre d'entrées

No : nombre de sorties

Wc : est définie par la formule mathématique :

$$Wc = \sum_{J=1}^{q} \left[\prod_{k=1}^{m} \sum_{I=1}^{n} Wc(j,k.i) \right]$$

[120] Qui a les valeurs suivantes: 1 pour *sequence* ; 2 pour *if then else* ; 3 pour *case*, et *while*. (Selon le Tableau C.2)

Les auteurs considèrent que « *Bien que la CFS soit une bonne mesure, elle ne fournit pas une bonne idée sur la quantité d'information contenue dans le logiciel* ». Après cette présentation, les auteurs introduisent la mesure CICM, qui est calculée à partir du poids de l'information contenue dans le logiciel et le poids cognitif des structures de contrôle de base du logiciel.

Il est important de souligner qu'ils reprennent la définition de logiciel de Wang (2004) qui est fort loin de celle classique d'IEEE Std 610.12 (1990). Pour Wang (2004) en informatique cognitive le logiciel est perçu comme : « *l'information de conception décrite de façon formelle et la mise en œuvre des instructions* ». La difficulté de compréhension du logiciel est donc étroitement liée à la difficulté de compréhension de cet ensemble d'informations.

Ils définissent donc l'information contenue dans une ligne de code comme une fonction d'identificateurs et d'opérateurs : I (k) = ($\sum$ des identificateurs + $\sum$ des opérateurs)

Et l'information contenue dans le logiciel en entier est définie comme la somme des informations contenues dans les lignes de code dudit logiciel : ICS = $\sum$ I (k) (k allant de 1 à LOCS), avec :

- I(k) : l'information contenue dans la kième ligne de code du logiciel, et
- LOCS : nombre total des lignes de code du logiciel

Le poids de l'information d'une ligne de code du logiciel est défini en fonction de la quantité d'information contenue dans le logiciel en entier (ICS) et les lignes de code : WICL (k) = ICS (k) / [LOCS – k], avec :

- K : le numéro d'une ligne de code du logiciel, et
- ICS (k) : l'information contenue dans le logiciel pour la kième ligne.

Le poids de l'information du logiciel est défini comme la somme des poids des informations des lignes de code : WICS = $\sum$ WICL (k) (k allant de 1 à LOCS).

En se basant sur les formules déjà citées, on peut maintenant définir la CICM, comme le produit du poids de l'information du logiciel et du poids cognitif des structures de contrôle de base : CICM = WICS * Wc

RÉFÉRENCES

Aho A., Sethi R. et Uilman J. (1986) Compilers: Principles, Techniques, and Tools. Addison-Wesley.

Allen Edward B., Khoshgoftaar Taghu M. et Chen Ye (2001) Measuring coupling and cohesion of software modules: An information-theory approach. Proceedings of the seventh International Software Metrics Symposium, London, p. 124-134.

Arisholm Eric et Briand Lionel C. (2004) Dynamic Coupling measurement for object-oriented software. IEEE Transactions on Software Engineering, Volume 30, Number 8, p. 491-506.

Basili V.R. et Hutchens D.H. (1980) A study of structural complexity metrics. Proceedings of the ACM/NBS Nineteenth Annual Technical Symposium; Pathways to System Integrity, p. 13-16.

Basili V.R. et Katz E.E. (1983) Metrics of interest in an Ada development. IEEE Workshop on Software Engineering Technology Transfer, pages 22-29, Miami, FL.

Basili V.R. et Weiss D.M. (1984) A methodology for collecting valid software engineering data. IEEE transactions on software engineering.

Basili Victor R. (1977) Evaluating Software Development Characteristics: Assessment of Software Measures in the Software Engineering Laboratory. Proceedings, 6th Software Engineering Workshop. NASA/Goddard Space Flight Center, IEEE Standard 729-1983, IEEE Standard Glossary of Software Engineering Terminology, New York: IEEE, 1983.38 pp.

Basili Victor R. et Zelkowitz Marvin V. (1979) Measuring software development characteristics in the local environment. Computer and Structures, Volume 10, p. 39-43.

Bell D. (2000) Software Engineering: A Programming Approach. Harlow: Pearson Educated LTD.

Berander P., Damm L.-O. , Eriksson J., Gorschek T., Henningsson K., Jonsson P., Kagstrom S., Milicic D., Martensson F., Ronkko K. et Tomaszewski P (2005) Software Quality Attributes and Trade-Offs. Suède: Lars Lundberg, Michael Mattsson, Claes Wohlin.

Bieman J. et Kang B.-K. (1995) Cohesion and Reuse in an Object-Oriented System. Proc. ACM Symp, Software Reusability, SSR'95, pp. 259–262.

Bieman J. et Kang B.-K. (1998) Measuring design-level cohesion. IEEE Transactions on Software Engineering (TSE), Volume 24, Number 2, p. 111-124.

Bieman J. et Ott L. (1994) Measuring Functional Cohesion. IEEE Transactions on Software Engineering, vol. 20, pp. 644-657.

Bieman J. et Ott Linda M. (1994) Measuring Functional Cohesion. IEEE Transactions on Software Engineering, Volume 20, Number 8, p. 644-657.

Boehm B. (1978) Characteristics of software quality. Vol 1 of TRW series on software technology, Amsterdam, Hollande.

Braude Eric J. (2001) Software Engineering: An Object-Oriented Perspective. États-unis: John Wiley & Sons, Inc.

Briand L.C., Daly J. W. et Wust J. K (1999) A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, 25(1):91–121.

Briand L.C., Morasca S. et Basili V.R. (1996) Property based software engineering measurement. IEEE Transactions on Software Engineering, 22(1): 68.

Briand Lionel C., Daly John W. et Wust Jurgen K. (1999) A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, Volume 25, Number 1, p. 91-121.

Briand Lionel C., Wust Jurgen et Lounis Hakim (1999) Using coupling measurement for impact analysis in object-oriented systems. IEEE International Conference Software Maintenance, p. 475-482.

Bunge M. (1977) Treatise on Basic Philosophy : Ontology I. The furniture of the World. Boston : Riedel.

Card D.N., Church V.E. et Agresti W.W. (1986) An empirical study of software design practices. IEEE Transactions on Software Engineering, 12 (2), 264-71.

Card David N., Page Gerald T. et McGarry Frank E. (1985) Criteria for software modularization. Proceedings of the 8th International Conference on Software Engineering, August 28-30, London, p. 372-377.

Chae H.S., Kwon Y.R. et Bae D.H. (2000) A Cohesion Measure for Object-Oriented Classes. Software Practice & Experience, Volume 30, Number 12, pp.1405-1431.

Chidamber Shyam R. et Kemerer Chris R. (1994) A metrics suite for object oriented design. IEEE Transactions on Software Engineering, Volume 20, Number 6, p. 476-493.

Darcy David P. et Kemerer Chris F (2002) Software complexity: Toward a unified theory of coupling and cohesion. MIS research Center, Carlson School of Management, University of Mannesota.

Darnas D.L. (1972) On the criteria to be used in decomposing systems into modules. Communications of the ACM, p. 1053-1058.

De Gruyter W. (1991) Software Complexity Measures and Methods. Berlin: Fevzi Belli and Hinrich E. Bonon.

Détienne F. (1998) Génie logiciel et psychologie de la programmation. Paris : Edition HERMES.

Dharmender, Singh Kushwaha, A. K. Misra. 2006. « A Modified Cognitive Information Complexity Measure of Software ». ACM SIGPLAN Software Engineering Notes, Volume 31, Number 7.

Dharmener, Singh Kushwaha, A. K. Misra. 2006. « Cognitive Complexity Metrics and its Impact on Software Reliability based on Cognitive Software Development Model ». ACM SIGGSOFT Software Engineering Notes, Volume 31, Number 2, p. 1-6.

Dijkstra Edsger W. (1968). A Case Against the GO TO Statement. Communications of the ACM, 3: 147–148.

Edward B. Allem et Khoshgoftaar Taghi M. (1999) Measuring coupling and cohesion: An information-theory approach. IEEE International Symposium on Software Metrics.

Emerson Thomas J. (1984) A discriminant metric for module cohesion. Proceedings of the Seventh International Conference of Software Engineering, Orlando, Florida, United States, p. 294-303.

Fenton N. E. et Pfleeger S. L. (1997) Software Metrics A rigorous & Practical Approach. London: Cmabridge Unversity Press, Cambridge.

Fenton N.E. et Melton A. (1990) Deriving structurally based software measures. Journal of Systems and Software, 12 (3): 177-87.

Gall H., Hajek K. et Jazayeri M. (1998) Detection of logical coupling based on product release history. Proceedings of the 14th International Conference on Software Maintenance (ICSM).

Ghezzi C., Jazayeri M. et Mandrioli D.(2003) Fundamentals of Software Engineering. Toronto: Pearson Education Canada Inc.

Halstead M. H. (1977) Elements of Software Science. New York: Elsevier North-Holland.

Henry S. et Kafura D. (1981) Software structure metrics based on information flow. IEEE Transactions on Software Engineering, 7 (5), 510-18.

IEEE Std 610.12 (1990) IEEE Standard Glossary of Software Engineering Terminology. Institute of Electrical and Electronics Engineers.

IEEE Std. 982.1 (1988) Standard Dictionary of Measures to Produce Reliable S/W. Institute of Electrical and Electronics Engineers.

IEEE Std. 982.2 (1988) Guide for the Use of IEEE Standard Dictionary of measures to produce Reliable S/W. Institute of Electrical and Electronics Engineers

IEEE Std.1061 (1998) IEEE Standard for a S/W Quality Metrics Methodology. Institute of Electrical and Electronics Engineers.

IEEE SWEBOK (2001) Guide to the Software Engineering Body of Knowledge. www.swebok.org

IEEE/EIA 12207 (1995) Software Life Cycle Processes - Life Cycle Data. Institute of Electrical and Electronics Engineers, Electronic Industries Association.

IEEE/EIA 12207.1 (1997) Guide for Information Technology Software life Cycle Processes—Life Cycle Data –Description. Institute of Electrical and Electronics Engineers, Electronic Industries Association.

ISO/IEC 9126-1 (2001) Software engineering -- Product quality -- Part 1: Quality model. International Organization for Standardization, International Electrotechnical Commission.

ISO/IEC TR 9126-2 (2003) Software engineering -- Product quality -- Part 2: External metrics. International Organization for Standardization, International Electrotechnical Commission.

ISO/IEC TR 9126-3 (2003) Software engineering -- Product quality -- Part 3: Internal metrics. International Organization for Standardization, International Electrotechnical Commission.

ISO/IEC TR 9126-4 (2004) Software engineering -- Product quality -- Part 4: Quality in use metrics. International Organization for Standardization, International Electrotechnical Commission.

Jordan S. (1997) Software metrics collection: Two new research tools. Master's thesis, Florida Atlantic University, oca Raton, FL. Advised by Taghi M. Khoshgoftaar.

Kitchenham B.A., Pfleeger S.L. et Fenton N.E. (1995) Towards a framework for software measurement validation. IEEE Transactions on Software Engineering, 21(12):929.

Klemola Tuomas et Rilling Juergen (2003) A Cognitive Complexity Metric Based on Category Learning. IEEE International Conference on Cognitive Informatics.

Knuth Donald E. (1977) Structured Programming With go to Satements. Current trends in Programming Methodology, Prentice-Hall.

Lakhotia Arun (1993) Rule-based approach to computing module cohesion. Proceedings of the 15th International Conference on Software Enginnering, May 17-21, Baltimore, p. 35-44.

Longworth H.D. (1985) Slice based program metrics. Master's thesis, Michigan Technological University.

Maffezzini Ivan, Martin Louis et Premiana Alice (2005). Prolégomènes à une critique du génie logiciel - Partie IV Exigences. Génie logiciel, no 72, p. 2-23.

Maffezzini, Premiana et Ventimiglia (2004) Prolégomènes à une critique du génie logiciel - Partie II : Qualité et mesures des produits. Génie Logiciel, Mars 2004, Numéro 68.

Martin Robert C. (1997) Granularity. Object Mentor.

Martin Robert C. (2000) Design Principles and Patterns. Object Mentor.

McCabe T. J. (1976) A complexity measure. IEEE Trans. Software Eng. vol. SE-2, No. 4, pp. 308-320.

McCall J.A., Richards P.K. et Walters G.F. (1977) Factors in software quality. Vols I-III, Rome Air Development Centre, Italie.

Mehra B. (1997) Measuring Data Cohesion in the Object-Oriented Paradigm. Master's thesis, Dept. of Computer Science, Michigan Technological Univ.

Meyers Timothy M. et Binkley David (2004) Slice-based cohesion metrics and software intervention. Proceedings of the 11th Working Conference on Reverse Engineering (WCRE), Delft University, the Netherlands 9-12 November.

Milicic D. (2005) Software quality attributes and trade-offs, chapter1 - Software Quality Models and Philosophies.

Miller G.A. (1956) The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychological Review, New York, vol. 63, pp. 81-87.

Mitchell R.J. (1990) Managing Complexity in Software Engineering. London: Peter Peregrinus LTD.

Myers G. (1978) Composite structured design. Van Nostrand Reinhold.

OQLF (Dernière consultation: Mai 2008) Office Québecois de la Langue Française. www.granddictionnaire.org

Ott L., Bieman J., Kang B.-K. et Mehra B. (1995). Developing Measures of Class Cohesion for Object-Oriented Software. Proc. Ann. Oregon Workshop Software Metrics, AOWSM'95.

Ott Linda M. et Thuss Jeffrey J. (1989) The relationship between slices and modules cohesion. Proceedings of the 11th International Conference on Software Engineering, p. 198-204.

Ott Linda M. et Thuss Jeffrey J. (1993) Slice based metrics for estimating cohesion. Proceedings of the First International Software Metrics Symposiums, May 21-22, Baltimore.

Page-Jones M. (1980) The Practical Guide to Structured Systems Design. Yourdon Press, New York, NY.

Pair C. (1993) Programming, Programming Languages and Programming Methods. Psychology of Programming, J.-M Hoc, T.R.G. Green, R. Samurcay, and F.J. Gilmore, Eds. New York: Academic Press, pp. 9-19.

Pfleeger S. Lawrence (2001). Software Engineering Theory and Practice. Toronto: Prentice-Hall Canada Inc.

Randall W. Jensen et Tonies Charles C. (1979) Software Engineering, Toronto: PRENTICE-HALL of Canada, LTD.

Roger S. Pressman (1982) Software Engineering A Practitioner's Approach. Toronto: McGraw-Hill Software Engineering and Technology.

Roger S. Pressman (1987) Software Engineering A Practitioner's Approach. Toronto: McGraw-Hill Book Company.

Roger S. Pressman (1992). Software Engineering A Practitioner's Approach. McGraw-Hill, Inc.

Roger S. Pressman (1997). Software Engineering A Practitioner's Approach. Toronto: McGraw-Hill Companies, Inc.

Roger S. Pressman (2005). Software Engineering A Practitioner's Approach. Toronto: McGraw-Hill Companies, Inc.

Schulmeyer G. Gordon (1999) Handbook of Software Quality Assurance. Prentice Hall.

Selbi R.W. et Basili V.R. (1988) Analyzing error-prone system coupling and cohesion. Technical Report, UMIACS-TR-88-46, Computer Science, University of California.

Shepperd M. (1990) Design metrics: an empirical analysis. Software Engineering Journal, 5 (1) 3-10.

Stevens W.P., Myers G.J. et Constantine L.L. (1974) Structured design. IBM Systems Journal, 13(2): 115–139.

TLFI (Dernière consultation: Juillet 2008) Trésor de la langue française informatisé. http://atilf.atilf.fr/tlf.htm

Tracz W.J. (1979) Computer Programming and the Human Thought Process. Software Practice and Experience, vol. 9, pp. 127-137.

Troy D.A. et Zweben S.H. (1981) Measuring the quality of structured design. Journal of Systems and Software, 2, 113-20.

Wang Y. (2004) On the Cognitive Informatics Foundations of Software Engineering. IEEE International Conference on Cognitive Informatics.

Wang Y. et Shao J. (2003) Measurement of the Cognitive Functional Complexity of Software. IEEE International Conference on Cognitive Informatics.

Weiser Mark (1981) Program slicing. Proceedings of the Fifth International Conference of Software Engineering, New York, p. 439-449.

Weyuker Elaine J. (1988) Evaluation Software Complexity Measure. IEEE Transactions on Software Engineering, Volume 14, Number 9, p. 1-4.

Wood R.E. (1986) Task Complexity: Definition of the Construct. OBHDP, vol. 37, pp. 60-82.

Woodward M.R. (1993) Difficulties using cohesion and coupling as quality indicators. Software Quality Journal, volume 2, number 2, p. 109-127.

Woodward M.R., Hennell M.A. et Hedley D. (1979) A measure of control flow complexity in program text. IEEE Trans. Software Eng., vol. SE-5, no. 1, pp. 45-50.

Yang Hong Yul et Berrigan Rebecca (2005) Detecting indirect coupling. Proceedings of the 2005 Australian Software Engineering Conference (ASWEC), p. 212-221.

Yourdon E. et Constantine L. (1979) Structured Design-Fundamentals of a Discipline of Computer Program and Systems Design. Toronto: Prentice-Hall of Canada, LTD.