EPM/RT-93/29

Analyse critique de la mesure du couplage logiciel

Denis Valois M. Sc.

Pierre N. Robillard, Ph.D., Ing.

Département de génie électrique
et génie informatique

École Polytechnique de Montréal

novembre 1993

Dépôt légal, novembre 1993
Bibliothèque nationale du Québec
Bibliothèque nationale du Canada

Pour se procurer une copie de ce document, s'adresser:

Les Éditions de l'École Polytechnique
École Polytechnique de Montréal
Case postale 6079, succ. Centre-ville
Montréal, (Québec) H3C 3A7
Téléphone: (514) 340-4473
Télécopie: (514) 340-3734

Compter 0.10 $ par page et ajouter 3,00 $ pour la couverture, les frais de poste et la manutention. Régler en dollars canadiens par chèque ou mandat-poste au nom de l'École Polytechnique de Montréal.

Nous n'honorerons que les commandes accompagnées d'un paiement, sauf s'il y a eu entente préalable dans le cas d'établissements d'enseignement, de sociétés ou d'organismes canadiens.

TABLE DES MATIÈRES

SOMMAIRE

Une bonne conception logicielle est à la base d'une programmation de qualité, de même qu'une bonne programmation facilite l'entretien, l'analyse inverse et la ré-utilisation. Au coeur même de ces concepts se retrouve la notion de "module". Dans plusieurs disciplines en génie logiciel on s'intéresse soit à identifier les caractéristiques d'une modularisation de qualité, pour éventuellement proposer des critères menant à une bonne conception, soit à mesurer la qualité d'une conception ou d'une programmation. Ainsi, la ré-utilisabilité d'un logiciel sera-t-elle en partie fonction de sa modularité. Malheureusement, aucun consensus n'est apparu dans ces disciplines, que ce soit sur des notions aussi élémentaire que la définition d'un module, et encore moins sur les critères pouvant servir à mesurer une modularisation. En effet, bien que le couplage soit défini comme les liens qui relient ensemble deux ou plusieurs modules, les critères de qualité du couplage sont subjectifs, et difficilement mesurables. D'autres parts, quelques métriques de couplage ont été proposées, mais les modèles sous-jacents, et les métriques, sont ad-hoc soit à un type d'application (ré-utilisabilité, analyse inverse, standards de programmation, etc.), soit à un environnement particulier (package Ada, procédure Pascal, objet C++, etc.).

Le but premier de cette recherche est donc de faire le point sur la mesure du couplage logiciel, tout d'abord en passant en revue critique les notions de module et de couplage, puis en présentant exhaustivement l'état-de-l'art de la mesure du couplage en insistant sur une présentation comparative de la force et de la faiblesse de chacun des modèles.

Une taxonomie des constructeurs modulaires est présentée à la section 1.3. En 2.3, un modèle de représentation et d'analyse du couplage est proposé. Ces deux items constituent un apport original à cette recherche.

Finalement, une problématique est identifiée et plusieurs hypothèses de recherche sont soumises.

REMERCIEMENTS

Je tiens à remercier certaines personnes étroitement liées à cette recherche.

En tout premier lieu, à mon directeur de recherche Pierre N. Robillard je présente mes respects et ma gratitude. Son talent pour poser les bonnes questions furent une source constante de motivation et de stimulation.

Denise Bigué, mon épouse, pour son infinie patience à mon égard ainsi que sa *cohésion* constante mérite une mention honorable, de même que mon *module-fils* Claude pour avoir *hérité* de certaines *fonctions*, et à l'encontre de toutes théories conserve un *couplage élevé* avec ses parents.

Je remercie également le Collège militaire de Saint-Jean pour l'environnement stimulant et le soutien financier.

INTRODUCTION

Le couplage logiciel est d'une grande importance en conception, en programmation et en analyse inverse. C'est un critère incontournable en qualité logicielle. Il est pourtant essentiel dans le contexte de ce document de définir le couplage et son domaine d'application, de présenter les corrélations entre le couplage et la qualité logicielle, et d'exhiber quelques applications.

Dans le cadre de cette étude, le couplage-système est différencié du couplage inter-modulaire. Le couplage inter-modulaire est constitué par les liens reliant entre eux deux ou plusieurs modules, alors que le couplage-système est défini comme l'ensemble du couplage inter-modulaire présent dans le système. Bien que ce soit, à la limite, un abus terminologique, cela permet de considérer deux systèmes en fonction de leur couplage. Il est universellement accepté qu'un bas couplage est préférable à un haut couplage; un bas couplage désigne peu de dépendances inter-modulaires et est généralement associé à la qualité logicielle, tandis qu'un haut couplage dénote beaucoup de dépendances inter-modulaires et implique la pauvreté logicielle [Sommerville-89, Schach-90].

Le couplage inter-modulaire est donc une abstraction englobant toutes les dépendances inter-modulaires. Il sera donc question de couplage-donnée et de couplage-contrôle. Le couplage-donnée est présent quand un module réfère (ou peut référer) une donnée déclarée ailleurs, que ce soit en lecture ou en modification. Dans ce dernier cas, il est d'usage de qualifier ce genre d'accès d'effet de bord. Le couplage-contrôle est identifié par l'invocation de l'exécution d'une routine déclarée extérieurement. Ce genre

de distinction peut paraître évident, mais elle est somme toute assez artificielle: dans quelques situations, la frontière entre le couplage-donnée et le couplage-contrôle est plutôt floue, comme dans l'invocation d'exécution d'une variable-procédure. La dépendance est à la fois une dépendance-contrôle et une dépendance-donnée. Une dépendance sur un objet contenant des méthodes est aussi un cas non évident. De plus, on peut aussi considérer le couplage-référence, caractérisé par une référence à un contenant sans référence au contenu, qu'il soit contrôle ou donnée. Etonnament, il est rare dans la littérature de distinguer ces trois cas.

Le domaine d'application du couplage est l'ensemble sur lequel on l'observe et sur lequel on espère le calculer: soit l'ensemble des modules constituant un système logiciel. Une modélisation du couplage est donc indissociable d'un modèle de la notion de module: la première partie en est consacrée. Ici, le terme "modularité" désigne une décomposition en modules effective, alors que "modularisation" réfère à l'action de décomposer ou de concevoir un logiciel en modules. Ainsi, un ingénieur logiciel effectue une modularisation lors d'une conception et observe la modularité d'un système dans un cadre d'une analyse inverse. Une modularisation de qualité est aussi intuitive que difficile à définir et à enseigner [Bailie-91]. Dans [Woodfield-81a], l'auteur conduit une étude empirique de la corrélation entre la modularité et la compréhension de programme, et conclue (à notre grande surprise!) que plus un programme est bien modularisé, meilleure est sa compréhension. Malheureusement, son étude n'inclue pas de programme modularisé à outrance.

La mesure du couplage est importante et utile. Depuis la publication d'une taxonomie du couplage dans [Myers-75], de

nombreux auteurs ont défini et expérimenté plusieurs métriques dans toutes sortes de contextes.

Le couplage inter-modulaire peut être ainsi associé trivialement à l'effort nécessaire pour extraire un module de son contexte, à le ré-utiliser, ou à le re-concevoir [Colbrook-89, Choi-90, Purtilo-90, Power-90]. Dans [Selby-88], l'auteur montre, dans un contexte d'entretien de logiciel, comment une erreur peut être localisée dans les modules à haut couplage-donnée.

Moins trivialement, une mesure du couplage peut aider lors-de la conception et de la programmation [Reynolds-84, Mitchell-88, Gomaa-89, Muller-90].

Une étude empirique sur la corrélation entre le type de couplage [Myers-75] et la facilité d'entretien démontre que le type de couplage présent (et non sa quantité) a peu d'importance sur la facilité d'entretien [Lohse-84]: la classification de Myers ne reflète donc pas toute la réalité. L'importance d'un nouveau modèle du couplage à la fois qualitatif, quantitatif et universel, est donc évidente [Ejiogu-90]. D'un tel modèle dérivera une métrique applicable dans plusieurs contextes.

1. LA MODULARITÉ DU LOGICIEL

Qu'est-ce qu'un module? Il y aura autant de réponses différentes que d'auteurs! On peut définir un module comme un découpage permettant aux humains de mieux comprendre une réalité complexe.

Dans ce chapître, la notion de module sera cernée: depuis les premiers auteurs jusqu'à un parcours des constructeurs modulaires disponibles dans les langages et environnements contemporains. Finalement, une taxonomie de la modularité permettra de capturer quelques aspects fondamentaux. Cette catégorisation est originale à la présente recherche.

Dans le contexte de ce document, le terme "modularité" désigne une décomposition en modules déjà effectuée, alors que "modularisation" réfère à l'action de décomposer ou de concevoir un logiciel en modules. Ainsi, un ingénieur logiciel effectue une modularisation lors d'une conception et observe la modularité d'un système dans un cadre d'une analyse inverse.

1.1 Concepts fondamentaux et auteurs classiques

Une des premières tentatives de définition revient à David Parnas [Parnas-72a]: un module est un ensemble d'états, accompagné de fonctions pour changer d'état, et de d'autres fonctions retournant une vision de l'état actuel. Bien que cela puisse sembler curieux à prime abord, sept mois plus tard, il change sa définition de tout au tout [Parnas-72b, Parnas-85]: un module est une affectation de travail, i.e. une tâche à faire. Rien de bizarre dans tout ceci; le contexte était très différent.

Quelques années plus tard, Stevens et Myers dans [Stevens-74, Myers-78] définissent un module par un ensemble d'énoncés satisfaisant trois conditions:

(1) c'est un ensemble fermé,
(2) il a le potentiel d'être invoqué par n'importe quel autre module,
(3) il a le potentiel d'être compilé séparément.

Déjà cela semble à la fois général et plus précis. Toutefois, considérer un module comme une entité exclusivement exécutable semble trop limitatif: les fichiers de déclarations et les packages Ada entre autre ne peuvent être considérés comme des modules.

Peu de temps après, Yourdon et Constantine [Yourdon-79] offrent une définition acceptable: un module est

(1) une séquence lexicalement contigüe
(2) d'énoncé de programme,
(3) bornée par des éléments délimiteurs et
(4) ayant un nom d'agrégat.

Les quatre conditions énumérées couvrent un très large domaine et c'est cette définition qui est acceptée un peu partout.

Notons pourtant que ces définitions n'en sont pas... Les auteurs définissent un module par ses qualités et non par son essence. La petite définition fournie en tête de chapître décrit l'essence du module et non son existence.

1.2 Constructeurs modulaires

Lors d'une modularisation, il n'est pas évident si la structure syntaxique du langage de conception affecte la prise de décision associée au processus de raffinement successif. Reynolds [Reynolds-90] présente un modèle de raffinement successif couplé à la structure grammaticale du langage de support. Une question naturelle serait de s'interroger sur la relation entre la modularisation en programmation et les primitives syntaxiques offertes par le langage de programmation. Intuitivement, tout indique que la même relation existe au niveau de la programmation.

Les constructeurs modulaires disponibles dans les langages de programmation contemporains seraient donc le "support" de la modularisation à ce niveau, un peu comme le langage naturel est le support de la pensée humaine. De manière équivalente, s'il est difficile de réfléchir à un sujet inexprimable, il devrait être autant difficile de coder un type de module dans un langage ne l'offrant pas en primitive. As-t-on déjà essayé de coder une co-routine ou un moniteur en Fortran ou en Pascal?

Les constructeurs de modules au sens de Yourdon et Constantine sont considérés.

Parmi les constructeurs modulaires universellement reconnus [MacLennan-87, Ghezzi-87, Teufel-91], on compte:

* la procédure et la fonction
* la macro-définition, incluant le fichier de déclarations
* le package Ada, le module au sens de Modula-2
* la classe et l'objet, au sens de SMALLTALK et C++
* la base de données

Les primitives "historiques" incluent:

* le paragraphe COBOL
* la routine (GOSUB) BASIC
* le BLOCK DATA au sens de Fortran

Un module dit environnemental est indépendant du langage de programmation et n'est relatif qu'à l'environnement de programmation:

* le fichier
* l'unité de compilation

Les constructeurs à la sémantique spécialisée sont:

* la tâche Ada
* le générique Ada (procédure et package)
* le moniteur de Hoare-

1.3 Une taxonomie de la modularité

Dans cette section, une taxonomie de la modularité logicielle est présentée.

Le tableau ci-après est une classification des différents constructeurs modulaires par granulité. Le critère de base est capturé par la dichotomie contenant-contenu. Un constructeur strictement contenant ne peut être déclaré localement à un autre constructeur. Un constructeur uniquement contenu n'a pas le potentiel de définition imbricable: il est strictement atomique.

GRANULITÉ MODULAIRE	CONSTRUCTEURS MODULAIRES
granulité large (contenant)	fichier unité de compilation
granulité moyenne (contenant et contenu)	objet C++ procédure/fonction à-la Pascal constructeur local récursif
granulité fine (contenu)	routine style BASIC paragraphe COBOL macro-définition constructeur non-récursif

Le niveau de granulité exprime aussi la capacité de regroupement du contenant par rapport à la fonctionalité du contenu. Ainsi, au niveau le plus élevé tous les constructeurs modulaires sont susceptibles de s'y retrouver (plus ou moins imbriqués), alors qu'au niveau le plus bas se retrouvent les constructeurs atomiques. Le niveau de granulité se définit par la déclaration du constructeur: une procédure Ada peut être de granulité moyenne si elle est déclarée localement ou de granulité large si elle est compilable séparément. En ce sens, une fonction "C" est de granulité large parce qu'elle est toujours compilable séparément (non imbricable), alors qu'une procédure/fonction Pascal est de granulité moyenne parce qu'elle est toujours déclarée localement. Ce concept de granulité est donc fortement relié aux notions de globalité et de localité.

2. LE COUPLAGE INTER-MODULAIRE

Le couplage inter-modulaire est constitué par les liens reliant entre eux deux ou plusieurs modules; c'est une abstraction englobant toutes les dépendances inter-modulaires, quel que soit leur type. Il est universellement accepté qu'un bas couplage est préférable à un couplage élevé; un bas couplage désigne peu de dépendances inter-modulaires et est généralement associé à la qualité logicielle, tandis qu'un haut couplage dénote beaucoup de dépendances inter-modulaires et implique la pauvreté logicielle [Sommerville-89, Schach-90].

Le couplage et la cohésion constituent des critères très importants de qualité [Krauskopf-90]. Parce qu'il sont applicables tôt dans le cycle de vie (dès la conception), ils peuvent être utilisés dans toutes les phases subséquentes, incluant la programmation, l'entretien, l'analyse inverse, etc.

Dans ce chapître, les notions de couplage et de cohésion seront mis en relation, différentes taxonomies du couplage sont présentées, puis une modélisation originale de la représentation et de l'analyse du couplage est développé.

2.1 Le couplage versus la cohésion

La cohésion est le terme par lequel est désigné les relations fonctionnelles intra-modulaires [Yourdon-79]. Un module hautement cohérent ne contient aucun élément étranger à sa fonctionnalité, alors qu'une cohésion de bas niveau indique que le module contient plusieurs instructions/déclarations n'ayant aucun rapport entre eux.

Le couplage et la cohésion sont intimement reliés. En effet, ils sont une indication de la qualité de la décomposition modulaire: la cohésion capture le degré avec lequel chaque module implante une seule abstraction, alors que le couplage identifie l'indépendance de chaque module. A une bonne conception modulaire est associée un bas couplage et une cohésion élevée. Inversement, si ces deux critères sont satisfaits, un système sera considéré de bonne qualité modulaire.

Ainsi, Murtagh [Murtagh-84, Murtagh-91] présente des algorithmes de restructuration de système pour réduire le couplage-contrôle. Dans un contexte d'entretien logiciel, Cimitile [Cimitile-90] propose un outil CASE de restructuration pour conserver un couplage-donnée de bas niveau. Selby [Selby-88] utilise le ratio couplage/cohésion pour prédire la localisation d'erreurs.

2.2 La catégorisation du couplage

Myers [Myers-75] a le premier proposé une taxonomie du couplage. Il distingue les six niveaux suivants (traduction libre):

(1) couplage par donnée (data coupling)
(2) couplage par structure (stamp coupling)
(3) couplage par contrôle (control coupling)
(4) couplage extérieur (external coupling)
(5) couplage par région (common coupling)
(6) couplage par contenu (content coupling)

L'ordre de présentation coïncide avec l'ordre de préférence: le niveau (1) étant le meilleur et le niveau (6) étant le pire. Il faut distinguer la terminologie utilisée dans ce

document (couplage-donnée vs couplage-contrôle) du vocabulaire de Myers. Le couplage (1) par donnée désigne une dépendance inter-modulaire restreinte aux paramètres procéduraux. Le couplage (2) fait référence au passage par paramètre d'une structure dont quelques composantes sont effectivement utilisées. Le couplage (3) est présent quand la logique interne d'un module est contrôlée par un paramètre. Le couplage (4) extérieur implique la présence de variables globales homogènes. Le couplage (5) par région indique la présence de variables communes hétérogènes. Finalement, le couplage (6) désigne l'altération directe de contrôle ou de données dans un module, ou quand deux modules partagent le même code [Yourdon-79]. Un bon exemple de couplage par contenu est la "routine" BASIC qui peut partager le même espace que le programme principal: il n'est pas nécessaire d'invoquer GOSUB pour exécuter le code de la routine.

Yourdon et Constantine [Yourdon-79] reprennent la classification de Myers pour n'en faire qu'une dimension dans leur modèle. Ils définissent quatre facteurs influençant la quantité de couplage:

(1) type de connection inter-modulaire
    (transfers de contrôle)
(2) complexité de l'interface
    (le nombre d'items transférés)
(3) type de flot d'information
    (grosso-modo les niveaux de Myers)
(4) moment auquel l'information est associée aux
    identificateurs

Le facteur (1) est un exemple de couplage-contrôle: on tient compte de la complexité inhérente aux exceptions, aux retours alternatifs, etc. Les facteurs (2) et (3) sont des critères de couplage-donnée. Le facteur (4) est un critère

de conception seulement. En effet, il s'agit de distinguer entre la compilation, l'édition des lien et l'exécution. Une information lue à l'exécution engendre un couplage moins élevé qu'une constante spécifiée dans le code.

Quelques années plus tard, Hammons et Dobbs [Hammons-85] reconnaissent les limitations du modèle de Myers dans un contexte Ada. Ils proposent deux nouveaux niveaux:

(1) couplage par définition
(2) couplage par package

Le couplage (1) par définition est présent quand un module réfère une définition (un type par exemple) extérieur. Le couplage (2) désigne quand deux modules réfèrent un troisième module, et réfèrent des définitions différentes du troisième. Il est ironique de constater que ces situations ne sont pas uniques à Ada; elles se présentent courramment en C, à travers la macro-expansion de fichiers #include. Le langage C existait déjà en 1975.

Nielsen [Nielsen-86] considère la catégorisation du couplage dans un contexte de tâches parallèles Ada. Il introduit un niveau supplémentaire à ceux de Myers et de Hammons:

(1) couplage par concurrence (concurrency coupling)

Un système concurrent est bassement couplé si les intéractions des tâches sont bien balancées, si l'attente active est minimisée et si les instructions effectuées dans les rendez-vous sont minimisées.

Embley et Woodfield [Embley-87, Embley-88] considèrent une catégorisation du couplage dans un contexte de type de donnée abstrait (TDA), implanté dans un package Ada, un module MODULA-2, un objet C++, etc. Ils identifient cinq types de couplage, présentés depuis le pire jusqu'au meilleur (traduction libre):

(1) couplage par malignité (surreptitious coupling)
(2) couplage par exploitation (covert coupling)
(3) couplage par visibilité (overt coupling)
(4) couplage par exportation (export coupling)
(5) couplage nil (nil coupling)

Le couplage (1) fait référence à un client utilisant la connaissance de l'implantation, mais sans l'accéder directement. Par exemple, si on implante un TDA "vecteur réel" avec une liste triée sur les index, un client pourrait invoquer un parcours du vecteur dans l'ordre du tri plutôt que dans l'ordre naturel de l'application. Le couplage (2) "exploite" l'implantation au même niveau syntaxique du TDA. En Pascal par exemple, la syntaxe ne permet pas de cacher l'implantation d'un TDA: les structures de données et les procédures sont globales au client. Ainsi, le langage ne peut empêcher un client d'exploiter un TDA. Le couplage (3) par visibilité indique qu'un client accède l'implantation du TDA (l'implantation est visible). Enfin, le couplage (4) par exportation se différencie du couplage par donnée et du couplage par définition/package, car un client déclare sa propre variable et invoque les opérations définie globalement. Le couplage (5) nil représente l'absence de couplage.

Finalement, Rising et Calliss [Rising-92] reprennent la discussion depuis le début. Leur article fait le point sur [Myers-78, Yourdon-79, Hammons-85, Embley-87, Embley-88]. Ils intègrent les niveaux développés par les auteurs précédents en une taxonomie échelonnée sur huit niveaux, spécifiés du meilleur au pire:

(1) couplage nil [Embley]
(2) couplage par package [Hammons]
(3) couplage par exportation [Embley]
(4) couplage par définition [Hammons]
(5) couplage par visibilité [Embley]
(6) couplage extérieur [Myers]
(7) couplage par malignité [Embley]
(8) couplage par exploitation [Embley]

Il est intéressant de noter que Rising n'est pas en accord avec Embley sur les gravités relatives de ses deux pires couplages et a permuté dans sa liste les places respectives des couplage par malignité et par exploitation.

2.3 Une modélisation du couplage

Dans cette section, un modèle de la représentation et de l'analyse du couplage est présentée, de façon à identifier qualitativement les types de couplage directement et indirectement présents entre toutes les paires de modules. Ce modèle est indépendant de la granulité et du type de module considéré. Il est aussi indépendant de la taxonomie du couplage. Le modèle s'applique donc aussi bien au couplage à-la Myers entre des procédures Pascal qu'au calcul de l'ordre de compilation de fichiers source Ada.

Il est donc nécessaire de fixer a-priori les sujets et les objets de l'analyse: les sujets étant déterminés par les modules d'un système alors qu'une taxonomie du couplage définit les objets. Ainsi, la granulité modulaire identifiée au chapître 1 permet de cerner les constructeurs modulaires pertinents. Une liste exhaustive des modules présents doit être calculée. Le choix arbitraire d'une définition du couplage, comme par exemple celle de Myers, ou plus simplement l'échelle nominale "(couplage-donnée,

couplage-contrôle, couplage-référence)" détermine le type de résultat obtenu.

Le modèle est défini par une matrice à deux dimensions. Les rangées et les colonnes sont indexées par les modules considérés. Chaque élément de la matrice est un ensemble sur les valeurs de couplage. L'analyse consiste à définir tous les ensembles dans la matrice.

	CIBLE			
SOURCE	module_1	module_2	...	module_n
module_1	{...}	{...}		{...}
module_2	{...}	{...}		{...}
.				
.				
.				
module_n	{...}	{...}		{...}

Un module SOURCE est dépendant du module CIBLE. L'analyste codifie la présence de dépendance directe entre le module_i et le module_j par un ensemble dont les éléments sont des valeurs de l'échelle du couplage considéré. Trivialement, un ensemble vide signifie qu'aucun couplage n'existe entre les deux modules tandis qu'un ensemble complet signifie que tous les types de couplage sont présents.

La nature même du couplage est directionnelle; cette matrice n'est donc pas nécessairement symétrique. De façon évidente, un module A dépendant d'un module B n'implique pas nécessairement que le module B est aussi dépendant de A. Cette dernière situation se réalise en présence de deux procédures mutuellement récursives, par exemple. Il est toutefois important de noter que la diagonale principale ne

doit contenir que des ensembles vides. En effet, même si un module est "auto-dépendant" (!?) -- procédure directement récursive par exemple -- le couplage est considéré nul.

Cette matrice contient une représentation du couplage direct entre toutes les paires de modules. L'automatisation de ce processus dépend de la calculabilité de l'échelle de couplage utilisée.

Pour obtenir la matrice du couplage direct et indirect, il suffit de calculer la fermeture transitive de la matrice initiale. En présence de n modules, ce calcule consomme O(n^3) opérations et peut se faire sur-place.

Cette dernière matrice offre une représentation complète des dépendances de chaque module (par rangée) et des dépendances dont chaque module est la cible (par colonne).

3. LA MESURE DU COUPLAGE

La mesure du couplage est importante. Elle est, avec la cohésion, une bonne indication de la qualité modulaire d'un système [Yourdon-79, Card-85]. Sellers [Sellers-92] montre que la métrique de McCabe, la complexité cyclomatique, est insensible à la modularisation. Dans [Emerson-84] est décrite une métrique discriminant la cohésion.

Une mesure du couplage pourrait être utilisée à plusieurs étapes du cycle de vie [Basili-80, Conte-86], incluant la conception en tout premier lieu. La quantification d'une conception n'est pas neuve. Myers, Yourdon et Constantine ont développé les notion de couplage et de cohésion dans ce but. Une mesure de couplage est donc désirable.

3.1 La mesure basée sur les modèles existants

Dans l'espoir de pouvoir calculer une métrique basée sur les modèles décrits précédemment, il faut investiguer la calculabilité des différents types de couplages.

Dans le formalisme de Myers, on peut vérifier statiquement le couplage par donnée et le couplage par contrôle. Par contre, en présence de pointeurs, une vérification statique du couplage par structure est impossible. Le couplage par région est impossible à distinguer du couplage extérieur: comment estimer l'homogénéité d'une variable globale?

Les quatre facteurs de Yourdon et Constantine souffrent aussi de difficultés. Il n'existe pas de liste exhaustive des types de transfers de contrôle: c'est dépendant du langage. Les problèmes des niveaux de Myers sont tous présents dans le facteur (3).

Comme Hammons suggère une extension aux niveaux de Myers, on ne peut pas non plus calculer son échelle.

Le niveau supplémentaire proposé par Nielsen est trop ambigu pour pouvoir le calculer statiquement: comment estimer si les tâches sont bien balancées et si l'attente active est minimale? (clairement non calculable)

Dans l'échelle d'Embley, les niveaux (3), (4) et (5) sont certainement calculables. Par contre, il est évident que les niveaux (1) et (2) ne le sont pas.

3.2 La représentation du couplage

Dans cette section, la littérature relative à la représentation du couplage est passée en revue, avec le but avoué de démontrer que le traitement automatique de l'information pertinente au couplage est possible.

Yau et Grabow [Yau-80] définissent un graphe dirigé hiérarchique basé sur le principe de "graphe récursif" associé à une base de donnée relationnelle. Leurs applications sont strictement dans un environnement Pascal.

Louise Moser [Moser-90] exhibe une représentation des dépendances-données et du flot de contrôle dans un contexte Ada, incluant rendez-vous, levée d'exception, capture d'exception, terminaison de tâche, importation et initialisation de package, etc. Le traitement du graphe peut être intra ou inter-modulaire.

Callahan, Carle, Hall et Kenedy [Callahan-90] généralisent un résultat de Barbara Ryder [Ryder-79] en construisant un multi-graphe permettant le traitement des procédures et des fonctions passées en paramètre.

Cimitile, DiLucca et Maresca [Cimitile-90] distinguent et traitent les dépendances inter-modulaires actuelles et potentielles. Les dépendances potentielles sont utiles pour empêcher de saturer le graphe.

Narayan Debnath [Debnath-90] produit une synthèse du graphe de flot de contrôle et du flot des données, sous le nom de "Generalized Program Graph".

Dietrich et Calliss [Dietrich-91] utilisent une base de données relationnelles pour définir des relations d'importation, d'exportation, d'héritage et de déclaration, le tout dans de multiples contextes très différents.

Harrold et Malloy [Harrold-91, Harrold-93] ont accentué leurs efforts sur les manipulations efficaces d'un graphe de contrôle et de donnée. Les informations obtenues sont strictement inter-modulaires.

3.3 Métriques existantes

Cette section est sûrement la plus importante du présent document. En effet, une revue exhaustive et critique de la littérature relative à la mesure du couplage est élaborée. Sur la fois des lacunes et faiblesses constatées, une problématique pourra être identifiée plus loin. La présentation respecte l'ordre chronologique de publication. Aucune classification par thème n'est tentée. Il est important de noter que sur les quinze références citées, seulement quatre proposent explicitement une modélisation du couplage; les autres utilisent une mesure (le plus souvent naïve) du couplage comme composante d'une métrique dérivée dans une application, ou valident une taxonomie particulière.

Il est étonnant de constater que les plus anciennes tentatives, à l'exception de deux seulement, sont récentes. Il s'agit donc d'un problème relativement ancien, mais dont le domaine de recherche est en pleine effervescence.

En 1979, Yourdon et Constantine [Yourdon-79] suggèrent que le couplage d'une conception structurée soit quantifié à l'aide de la mesure du fan-in et du fan-out des composantes. Une valeur de fan-in élevée est associée à un couplage élevé parce que c'est une mesure directe de dépendance-contrôle. Une valeur de fan-out élevée est associée à une complexité élevée du module, en raison de la logique requise pour contrôler les invocations. Dans ce contexte, seule une mesure de couplage d'une conception de système est considéré.

Un peu plus tard, Henry et Kafura [Henry-81] proposent la mesure de couplage du module A au module B suivante:

couplage = (le nombre de procédures exportant de
l'information du module A
+
le nombre de procédures important de
l'information dans le module B)
*
le nombre de flots différents d'information.

Les auteurs soulignent qu'ils n'ont pas été capable de valider cette mesure. Il s'agit strictement d'une mesure de couplage-donnée.

Dans un domaine particulier (Ada), [Kirchgassner-87] présente un outil automatisant l'identification de regroupement modulaires et la hiérarchisation statique des modules. Le modèle de couplage est particulier à Ada (comme par exemple l'instantiation générique), et les relations de couplage sont différentes du niveau microscopique (module) au niveau macroscopique (groupement modulaire).

Selby et Basili [Selby-88] utilisent le ratio couplage/cohésion pour prédire l'effort d'entretien d'un très grand système. Leur modèle de couplage est simple. Le triplet *(p,x,q)* est appellé "couplage-donnée" (data-binding) si le module *p* communique avec le module *q* via la variable *x*. Le couplage entre *p* et *q* est le nombre de triplets *(p,x,q)*. Même si cette mesure paraît odieusement simple (un seul aspect du couplage à-la Myers), les auteurs ont pu la valider (eh oui!) sur un système d'environ 135 KLOC. En fait, les dés étaient pipés un peu: le système utilisé pour

la validation ne contenait que ce type de couplage-donnée. Par contre, Selby et Basili ont atteint de bons résultats en ignorant complètement le couplage-contrôle.

Encore une fois appliqué dans le monde Ada, Embley et Woodfield [Embley-88] posent le postulat a-priori qu'un package Ada ne doit contenir qu'une seule implantation de type de donné abstrait (TDA), et n'exporter que les opérations définies par ce TDA. Sous cette hypothèse, les auteurs ont trouvé que les packages n'ayant aucune connaissance de l'implantation des autres TDAs ont un couplage plus bas que les packages manipulant la structure de donnée d'un TDA. Cet article présente une validation de la taxonomie du couplage présenté dans [Embley-87].

Gopal et Schach [Gopal-89] présentent un outil CASE orienté Ada, permettant de retracer les références et les modifications de variables dans un contexte d'aide à l'entretien. Dans le même esprit que [Kirchgassner-87], cet outil permet aussi d'identifier les modules invoquants et les modules invoqués. La notion de couplage n'est pas explicitement discutée; elle est implicite en ce sens que l'outil est un microscope sur les liens inter-modulaires, autant statiques que dynamiquement observés. C'est un générateur de références croisées sophistiqué. Vanek et Culp [Vanek-89], Maarek [Maarek-88] présentent le même type d'outil, mais indépendant du langage. De même, [Robillard-91] exhibe un outil CASE sophistiqué qui intègre le calcul de plusieurs métriques. Ince [Ince-90b] utilise ce type d'outil pour détecter la dégradation structurelle lors de l'entretien.

Le couplage dans [Oval-89] est utilisé pour mesurer l'indépendance du graphe d'appel. Les arbres sous-jacents au graphe d'appel, dont les racines sont les points d'entrée

des applications, peuvent partager des modules de services. Ces modules sont qualifiés "indépendants". Les auteurs définissent plusieurs métriques sur une échelle ratio basée sur une mesure triviale du couplage-contrôle: les degrés d'incidence et d'excidence des noeuds (i.e. le nombre de modules distincts appelants et appelés). Ces métriques permettent l'identification des noeuds indépendants. Une critique peut sembler sévère: ce même résultat (l'identification des noeuds indépendants) peut se calculer directement sur le graphe d'appel par une fouille en profondeur suivie d'un parcours préfixé.

Adamov et Richter [Adamov-90] définissent la complexité "structurelle" inter-modulaire comprenant (entre autres) une complexité de flot-de-contrôle, qui est proche mais différente du couplage-contrôle (les modules directement récursifs sont reconnus et une "dépendance" du module vers lui-même est introduite, alors qu'il est évident que la récursivité directe n'engendre aucun couplage supplémentaire), et qui définit la complexité d'"interface", qui est essentiellement une tentative de couplage-donnée (en effet, le modèle indique que la "quantité" de dépendance-donnée est considérée, mais sans distinguer entre le nombre de variables référencées et la taille de la variable référencée: le problème de la "quantité" de couplage-donnée via un pointeur est escamoté). Il s'agit en fait d'une mesure topologique sur le graphe d'appel et sur le graphe d'interface.

Yaung et Raz [Yaung-92] analysent et mesurent des liens inter-processus générés par une conception via un diagramme de flot-de-données. Le couplage-contrôle est totalement absent. Le couplage-donnée est modélisé par une matrice d'interconnectivité. La mesure du couplage est une mesure de type "ratio". Une analyse de groupement est effectuée.

Dans [Cherniack-93], le couplage est modélisé par une combinaison linéaire (pondérée) des huit termes suivants:

- nombre de paramètres-donnée en mode IN,
- nombre de paramètres-contrôle en mode IN,
- nombre de paramètres-donnée en mode OUT,
- nombre de paramètres-contrôle en mode OUT,
- nombre de variables globales utilisées en donnée,
- nombre de variables globales utilisées en contrôle,
- nombre de modules invoqués,
- nombre de modules invoquant.

Les coefficients de la combinaison linéaire (la pondération de chaque terme) ont été déterminés empiriquement dans des contextes de ré-utilisabilité, de portabilité et d'entretien. Les auteurs ont tenté de calculer le couplage à-la Myers, mais restent silencieux sur la distinction entre les paramètres/variables-donnée et contrôle.

4. PROBLÉMATIQUE

L'expérience de Lohse [Lohse-84] et les travaux de Hammons, Nielsen et Embley [Hammons-85, Nielsen-86, Embley-87] ont démontré que les taxonomies du couplage élaborées par Myers/Yourdon-Constantine [Myers-75, Yourdon-79] ne reflètent pas toute la réalité. De plus, le modèle enrichi Yourdon-Hammons-Nielsen-Embley est intuitif, informel, et impossible à calculer: il décrit plus une complexité psychologique qu'une complexité logicielle objective. A ce même modèle enrichi correspond une échelle de type ordinal [Roberts-79, Kaposi-91, Zuse-91] qui ne permet pas d'établir de métriques comparatives. Finalement, dans la section 3.3, il a été démontré qu'il n'y a pas de modèle satisfaisant du couplage.

Le problème évident est de définir un modèle du couplage auquel correspondrait une échelle de type ordinal, intervalle, ratio ou idéalement de type absolu [Roberts-79, Kaposi-91, Zuse-91]; ce qui permet de définir une mesure objective. En fait, Fenton et Melton [Fenton-90] montrent comment associer une mesure du couplage basée sur le modèle de Myers à une échelle ordinale.

Cette mesure devrait être sensible au couplage-contrôle ET au couplage-donnée; elle serait potentiellement multi-dimensionnelle [Shepperd-90]. Elle serait vérifiée face aux axiomes de [Weyuker-88, Lakshmanan-91, Cherniavsky-91]. Récemment dans [Chung-91], la complexité est exprimée avec une notation asymptotique polynomiale, exactement comme une complexité-temps algorithmique.

A la lumière des chapîtres précédents, au moins trois pistes de recherche méritent une attention. Par ordre de préférence d'investigation:

HYPOTHÈSE 1: Un modèle du couplage serait dérivé des modèles d'interconnection basés sur la théorie de l'information [VanEmden-70, Chanon-74, Chen-78, Mohanty-81, Boloix-85, Boloix-88, Chapin-89, Robillard-89, Como-90, Torres-91, Harrison-92, Cook-93]. Une métrique obtenue reflèterait ainsi l'entropie du couplage. Une investigation au niveau inter-modulaire de la mesure (intra-modulaire) de Boloix semble prometteuse.

HYPOTHÈSE 2: Un modèle du couplage serait dérivé du modèle de découpage de programme ("program slicing") [Lyle-88, Ott-89, Horwitz-90].

HYPOTHÈSE 3: Un modèle du couplage serait dérivé de la représentation de programme par un "polynôme caractéristique" [Cantona-83].

Dans l'espoir de pouvoir calculer une éventuelle métrique, il est nécessaire de passer en revue les points suivants:

(1) un modèle de modularité assez expressif est-il disponible?
(2) est-il possible d'identifier (facilement) les sources potentielles de couplage-donnée et de couplage-contrôle pour tous les type de modules?
(3) une représentation du couplage est-elle disponible?
(4) existe-t-il des algorithmes efficaces pour manipuler cette représentation?
(5) le contexte d'utilisation de la métrique est-il bien défini?

La notion de module de Yourdon et Constantine est assez riche pour englober tous les constructeurs modulaires connus. La taxonomie proposée pour la modularité (chapître 1) ainsi que les caractérisations et la modélisation du couplage (chapître 2) permettent d'identifier les source potentielles pour chaque module.

Comme le couplage est typiquement représenté par un graphe dirigé ou un multi-graphe (c.f. section 3.2), les hypothèses 1, 2 et 3 sont équivalentes à associer la théorie de l'information, le découpage et l'algèbre polynomiale respectivement à la théorie des graphes. Idéalement, une éventuelle métrique serait calculée rapidement. Heureusement, plusieurs algorithmes efficaces relatifs au calcul de métriques ont été publiés à ce jour [Ryder-79, Overstreet-88, Mizuno-89, Muller-89, Ryder-90, Gusfield-91, Ammarguellat-92].

De plus, il faut définir le contexte d'utilisation d'une métrique de façon à effectuer une validation [Ejiogu-93] et conduire d'éventuelles expériences empiriques. Les contextes d'entretien, de ré-utilisation et plus génériquement de qualité de programmation sont également pertinents.

BIBLIOGRAPHIE

[AbdElHafiz-89]
Abd-El-Hafiz, S. K.; Basili, V. R.; Caldiera, G.
"Toward Automated Support for Extraction of Reusable Components",
Proceedings 1991 IEEE Conference on Software Maintenance, Sorrento Italy (Oct 1991), pp 212-219.

[Adamov-90]
Adamov, R.; Richter, L.
"A Proposal for Measuring the Structural Complexity of Programs",
Journal of Systems and Software,
vol 12 no 1 (April 1990), pp 55-70.

[Agnarsson-85]
Agnarsson, S.; Krishnamoorthy, M. S.
"Towards a Theory of Packages",
Proceedings ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments, Seattle Wa (June 1985),
in ACM SIGPLAN Notices, vol 20 no 7 (July 1985),
pp 117-130.

[Ammarguellat-92]
Ammarguellat, Zahira.
"A Control-Flow Normalization Algorithm and Its Complexity",
IEEE Transaction on Software Engineering,
vol 18 no 3 (March 1992), pp 237-251.

[Bailie-91]
Bailie, F.K.
"Improving the Modularization Ability of Novice Programmers",
ACM SIGCSE Bulletin, vol 23 no 1 (March 1991), pp 277-282.

[Baker-79]
Baker, A. L.; Zweben, S. H.
"The Use of Software Science in Evaluating Modularity Concepts",
IEEE Transactions on Software Engineering, vol 5 no 2 (1979), pp 110-120.

[Baker-80]
Baker, A. L.; Zweben, S. H.
"A Comparison of Measures of Control Flow Complexity",
IEEE Transactions on Software Engineering, vol SE-6 no 6 (Nov 1980), pp 506-512.

[Basili-80]
Basili, Victor R.
TUTORIAL ON MODELS AND METRICS FOR SOFTWARE MANAGEMENT AND ENGINEERING,
IEEE Computer Society Press, 1980.

[Bastani-87]
Bastani, Farokh B.; Iyengar, S. Sitharama.
"The Effect of Data Structures on the Logical Complexity of Programs",
Communications of the ACM, vol 30 no 3 (March 1987), pp 250-259.

[Beane-84]
Beane, J.; Giddings, N.; Silverman, J.
"Quantifying Software Designs",
Proceedings 7th International Conference on Software Engineering, Orlando USA (March 1984), pp 314-322.

[Belady-76]
Belady, L. A.; Lehman, M. M.
"A Model of Large Program Development",
IBM Systems Journal, vol 15 no 3 (1976), pp 225-252.

[Benedusi-89]
Benedusi, P.; Cimitile, A.; De Carlini, U.
"A Reverse Engineering Methodology to Reconstruct Hierarchical Data Flow Diagrams for Software Maintenance",
Proceedings 1989 IEEE Conference on Software Maintenance, Miami Florida (Oct 1989), pp 180-189.

[Bergstra-90]
Bergstra, J. A.; Heering, J.; Klint, P.
"Module Algebra",
Journal of the Association for Computing Machinery, vol 37 no 4 (April 1990), pp 335-372.

[Berlinger-80]
Berlinger, E.
"An Information Theory Based Complexity Measure",
Proceedings 1980 Nat. Computer Conf., pp 773-779.

[Boloix-85]
Boloix, Germinal.
MESURE DE LA COMPLEXITE DU LOGICIEL UTILISANT UN MODELE D'INTERCONNEXIONS,
Thèse, Ecole Polytechnique de Montréal, Juin 1985.

[Boloix-88]
Boloix, Germinal; Robillard, Pierre N.
"Interconnectivity Metric for Software Complexity",
INFOR, vol 26 no 1 (Feb 1988), pp 17-39.

[Bourdoncle-90]
Bourdoncle, F.
"Interprocedural Abstract Interpretation of Block Structured Languages with Nested Procedures, Aliasing and Recursivity",
Proceedings 1990 International Workshop on Programming Language Implementation and Logic, Linkoping Sweden (Aug 1990), pp 307-323.

[Bradley-91]
Bradley, L.
"Evaluating Complex Properties of Object-Oriented Design and Code",
Proceedings International Software Quality Conference, Dayton Ohio (Oct 1991), pp 32-36.

[Callahan-90]
Callahan, David; Carle, Alan; Wolcott Hall, Mary; Kennedy, Ken.
"Constructing the Procedure Call Multigraph",
IEEE Transactions on Software Engineering,
vol 16 no 4 (April 1990), pp 483-487.

[Calliss-89a]
Calliss, Frank W.
Inter-Module Code Analysis Techniques for Software,
Thèse, Durham University UK, 1989.

[Calliss-89b]
Calliss, F.W.; Cornelius, B.J.
"Two Module Factoring Techniques",
Journal of Sofware Maintenance: Research and Practice,
vol 1 no 2 (Dec 1989), pp 81-89.

[Calliss-90]
Calliss, Frank W.; Cornelius, Barry J.
"Potpourri Module Detection",
Proceedings 1990 IEEE Conference on Software
Maintenance, San Diego Calif. (Nov 1990), pp 46-51.

[Cantona-83]
Cantona, G.; Cimitile, A.; Sansone, L.
"Complexity in Program-Schemes: The Characteristic
Polynomial",
ACM SIGPLAN Notices, vol 18 no 3 (1983), pp 22-30.

[Caplinger-85]
Caplinger, Michael.
"Structured Editor Support for Modularity and Data
Abstraction",
Proceedings ACM SIGPLAN 85 Symposium on Language Issues
in Programming Environments, Seattle Wa (June 1985),
in ACM SIGPLAN Notices, vol 20 no 7 (July 1985),
pp 140-147.

[Card-85]
Card, D. N.; Page, G. T.; McGarry, F.E.
"Criteria for Software Modularization",
Proceedings 8th International Conference on Software
Engineering, London UK (Aug 1985), pp 372-377.

[Card-88]
Card D. N.; Agresti, W. W.
"Measuring Software Design Complexity",
The Journal of Systems and Software, no 8 (1988),
pp 185-197.

[Card-90]
Card, D. N.; Glass, R. L.
Measuring Software Design Quality,
Prentice-Hall, 1990.

[Carroll-88]
Carroll, M.D.; Ryder, B.G.
"Incremental Data Flow Analysis via Dominator and
Attribute Updates",
Proceedings 15th Annual ACM Symposium on Principles of
Programming Languages, San Diego Calif. (Jan 1988),
pp 274-284.

[Chanon-74]
Chanon, R. N.
"On a Measure of Program Structure",
Proceedings of the Programming Languages Symposium,
G. Goos and J. Hartmanis eds., Springer-Verlag,
Paris 1974, pp 9-16.

[Chapin-78]
Chapin, N.; Denniston, S. P.
"Characteristics of a Structured Program",
ACM SIGPLAN Notices, vol 13 no 5 (1978), pp 36-45.

[Chapin-89]
Chapin, N.
"An Entropy Metric for Software Maintainability",
Proceedings 22nd Annual Hawaii International Conference
on System Science vol II: Software Track,
Kailua-Kona Hawaii Jan 1989,
IEEE Computer Society Press, pp 522-523.

[Chen-78]
Chen, E. T.
"Program Complexity and Programmer Productivity",
IEEE Transactions on Software Engineering,
vol SE-4 no 3 (May 1978), pp 187-194.

[Cheng-91]
Cheng, C. K.; Yao, S. Z.; Hu, T. C.
"The Orientation of Modules Based on Graph
Decomposition",
IEEE Transactions on Computers, vol 40 (June 1991),
pp 774-780.

[Cherniack-93]
Cherniack, Jerome R.; Dhama, Harpal S.; Fandozzi,
Jeanne F.
"Tool for Computing Cohesion and Coupling in Ada
Programs: DIANA Dependent Part",
Proceedings 12th Ada-Europe International Conference,
Paris France June 1993, Springer-Verlag, pp 180-196.

[Cherniavsky-91]
Cherniavsky, John C.; Smith, Carl H.
"On Weyuker's Axioms for Software Complexity Measures",
IEEE Transactions on Software Engineering,
vol 17 (June 1991), pp 636-638.

[Choi-90]
Choi, S.C.; Scacchi, W.
"Extracting and Restructuring the Design of Large Systems",
IEEE Software, vol 7 no 1 (Jan 1990), pp 66-71.

[Chung-91]
Chung, C. M.; Lee, M. C.
"Polynomial Metric",
International Journal of Mini and Microcomputers,
vol 13 no 3 (1991), pp 89-99.

[Cimitile-90]
Cimitile, A.; Di Lucca G. A.; Maresca P.
"Maintenance and Intermodular Dependencies in Pascal Environment",
Proceedings 1990 IEEE Conference on Software Maintenance, San Diego Calif. (Nov 1990), pp 72-83.

[Cimitile-91]
Cimitile, Aniello; De Carlini, Ugo.
"Reverse Engineering: Algorithms for Program Graph Production",
Software Practice & Experience,
vol 21 (May 1991), pp 519-537.

[Colbrook-89]
Colbrook A.; Smythe C.
"The Retrospective Introduction of Abstraction into Software",
Proceedings 1989 IEEE Conference on Software Maintenance, Miami Florida (Oct 1989), pp 166-173.

[Como-90]
Como, G.; Lanubile, F.; Visaggio, G.
"Evaluation of characteristics of design quality metrics",
Proceedings 2nd International Conference on Software Engineering and Knowledge Engineering, Skokie IL USA (June 1990), pp 195-201.

[Conte-86]
Conte, S. D.; Dunsmore, H. E.; Shen, V. Y.
SOFTWARE ENGINEERING METRICS AND MODELS,
Benjamin/Cummings Publishing, 1986.

[Cook-82a]
Cook, M. L.
"Software Metrics: an Introduction and Annotated Bibliography",
ACM SIGSOFT Software Engineering Notes,
vol 7 no 2 (April 1982), pp 41-60.

[Cook-82b]
Cook, R. P.; Lee, I.
"A Contextual Analysis of Pascal Programs",
Software Practice and Experience, vol 12 (1982),
pp 195-203.

[Cook-91]
Cook, C.
"Information Theory Metric for Assembly Language",
Proceedings Third Annual Oregon Workshop on Software Metrics, March 1991.

[Cook-93]
Cook, Curtis R.
"Information Theory Metric for a Program Language",
Software Engineering Strategies,
vol 1 no 1 (March/April 1993), pp 52-60.

[Crookes-83]
Crookes, D.; Fee, R.; Pickering, V.
"Building Syntax Graphs from Syntax Equations:
A Case Study in Modular Programming",
Software Practice and Experience, vol 13 (Dec 1983),
pp 1129-1139.

[Debnath-90]
Debnath, N.C.
"A Study of Control Flow and Data Dependency
Interface",
Proceedings 18th ACM Annual Computer Science
Conference, Washington DC (Feb 1990), p 435.

[Dietrich-91]
Dietrich, Suzanne W.; Caliss, Frank W.
"The Application of Deductive Databases to Inter-Module
Code Analysis",
Proceedings 1991 IEEE Conference on Software
Maintenance, Sorrento Italy (Oct 1991), pp 120-128.

[Dumke-92]
Dumke, R.
Software Metrics:
A bibliography of our analysed books and papers,
Technical Report, University of Magdeburg, Germany,
March 1992.

[Ehrig-89a]
Ehrig, H.; Fey, W.; Hansen, H.; Lowe, M.; Jacobs, D.
"Algebraic Software Development Concepts for Module and Configuration Families",
Proceedings 9th Conference on Foundations of Software Technology and Theoretical Computer Science,
Bengalore India (Dec 1989), pp 181-192.

[Ehrig-89b]
Ehrig, H.; Fey, W.; Hansen, H.; Lowe, M.; Jacobs, D.; Langen, A.; Parisi-Presicce, F.
"Algebraic Specification of Modules and Configuration Families",
Journal of Information Processing and Cybernetics,
vol 25 no 5-6 (1989), pp 205-232.

[Ejiogu-90]
Ejiogu, Lem O.
"Beyond Structured Programming: an Introduction to the Principles of Applied Software Metrics",
Structured Programming, vol 11 no 1 (1990), pp 27-43.

[Ejiogu-91]
Ejiogu, Lem O.
Software Engineering with Formal Metrics,
QED Publishing Group, Wellesley, Ma, 1991.

[Ejiogu-93]
Ejiogu, Lem O.
"Five Principles for the Formal Validation of Models of Software Metrics",
ACM SIGPLAN Notices, vol 28 no 8 (August 1993),
pp 67-76.

[Elshoff-84]
Elshoff, J. L.
"Characteristic Program Complexity Measures",
Proceedings IEEE 7th International Conference on
Software Engineering, Orlando Florida (March 1984),
pp 288-293.

[Embley-87]
Embley, David W.; Woodfield, Scott N.
"Cohesion and Coupling for Abstract Data Types",
Proceedings 6th Annual International Phoenix Conference
on Computers and Communications, Phoenix Arizona
(1987), IEEE Computer Society Press, pp 229-234.

[Embley-88]
Embley, David W.; Woodfield, Scott N.
"Assessing the Quality of Abstract Data Types Written
in Ada",
Proceedings 10th International Conference on Software
Engineering, 1988, IEEE Computer Society Press,
pp 144-153.

[Emerson-84]
Emerson, T. J.
"A Discriminant Metric for Module Cohesion",
Proceedings 7th International Conference on Software
Engineering, Orlando Florida (March 1984), pp 294-303.

[Engberts-91]
Engberts, Andre; Kozaczynski, Wojtek; Ning, Jim.
"Concept Recognition-Based Program Transformation",
Proceedings 1991 IEEE Conference on Software
Maintenance, Sorrento Italy (Oct 1991), pp 73-82.

[Esteva-90a]
Esteva, J. Carlo.
"Learning to Recognize Reusable Software by Induction",
Proceedings SPIE - The International Society for
Optical Engineering, vol 1293 no pt.2 (1990),
pp 654-670.

[Esteva-90b]
Esteva, J. C.; Reynolds, R. G.
"Learning to Recognize Reusable Software by Induction",
Proceedings 2nd International Conference on Software
Engineering and Knowledge Engineering,
Skokie IL (June 1990), pp 19-24.

[Evangelist-83]
Evangelist, W. M.
"Software Complexity Metric Sensitivity to Program
Structuring Rules",
The Journal of Systems and Software, vol 3 (1983),
pp 231-243.

[Fenton-86]
Fenton, N. E.; Whitty, R. W.
"Axiomatic Approach to Software Metrication Through
Program Decomposition",
The Computer Journal, vol 29 (Aug 1986), pp 330-339.

[Fenton-87]
Fenton, Norman E.; Kaposi, Agnes A.
"Metrics and Software Structure",
Information and Software Technology,
vol 29 (July/Aug 1987), pp 301-320.

[Fenton-90]
Fenton, N.; Melton, A.
"Deriving Structurally Based Software Measures",
Journal of Systems and Software,
vol 12 no 3 (July 1990), pp 177-187.

[Gannon-87]
Gannon, J. D.; Hamlet, R. G.; Mills, M. D.
"Theory of Modules",
IEEE Transactions on Software Engineering,
vol SE-13 no 7 (July 1987), pp 820-829.

[George-85]
George, Geoffrey; Leathrum, James F.
"Orthogonality of Concerns in Module Closure",
Software Practice and Experience,
vol 15 no 2 (Feb 1985), pp 119-130.

[Ghezzi-87]
Ghezzi, Carlo; Jazayeri, Mehdi.
Programming Language Concepts,
John Wiley & Sons, New York, 1987.

[Gill-91]
Gill, Geoffrey K.; Kemerer, Chris F.
"Cyclomatic Complexity Density and Software Maintenance Productivity",
IEEE Transactions on Software Engineering,
vol 17 no 12 (Dec 1991), pp 1284-1288.

[Gomaa-89]
Gomaa, H.
"Structuring Criteria for Real-Time System Design",
Proceedings 11th International Conference on Software Engineering, Pittsburg PA (May 1989), pp 290-301.

[Gopal-89]
Gopal, Rajeev; Schach, Stephen R.
"Using Automatic Program Decomposition Techniques in Software Maintenance Tools",
Proceedings 1990 IEEE Conference on Software Maintenance, Miami Florida (Oct 1989), pp 132-141.

[Gordon-79]
Gordon, R. D.
"Measuring Improvements in Program Clarity",
IEEE Transactions on Software Engineering, vol SE-5 no 2 (March 1979), pp 79-90.

[Gries-85]
Gries, David; Prins, Jan.
"A New Notion of Encapsulation",
Proceedings ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments, Seattle Wa (June 1985), in ACM SIGPLAN Notices, vol 20 no 7 (July 1985), pp 131-139.

[Gusfield-91]
Gusfield, Dan.
"Computing the Strength of a Graph",
SIAM Journal on Computing, vol 20 no 4 (Aug 1991), pp 639-654.

[Hall-83]
Hall, N. R.; Preiser, S.
"Dynamic Complexity Measures for Software Design",
IEEE Computer, (1983), pp 57-66.

[Hall-84]
Hall, N. R.; Preiser, S.
"Combined Network Complexity Measures",
IBM Journal of Research and Development,
vol 28 no 1 (Jan 1984), pp 15-27.

[Hall-93]
Hall, Mary W.; Kennedy, Ken.
"Efficient Call Graph Analysis",
ACM Letters on Programming Languages and Systems,
vol 1 no 3 (Sept 1993), pp 227-242.

[Hammons-85]
Hammons, Charles; Dobbs, Paul.
"Coupling, Cohesion and Package Unity in Ada",
ACM SIGAda Ada Letters, vol 4 no 6 (1985), pp 49-59.

[Hansen-88]
Hansen, H.; Lowe, M.
"Modular Algebraic Specifications",
Proceedings International Workshop on Algebraic and
Logic Programming, Gaussig East Germany (Nov 1988),
pp 168-179.

[Harrison-84]
Harrison, W.
"Bibliography on Software Complexity Metrics",
ACM SIGPLAN Notices, vol 19 no 2 (1984), pp 17-27.

[Harrison-87]
Harrison, W.; Cook, C. R.
"A Micro/Macro Measure of Software Complexity",
The Journal of Systems and Software, vol 7 (1987),
pp 213-219.

[Harrison-92]
Harrison, W.
"An Entropy-Based Measure of Software Complexity",
IEEE Transactions on Software Engineering,
vol 18 no 11 (Nov 1992), pp 1025-1029.

[Harrold-89]
Harrold, M.J.; Soffa, M.L.
"Interprocedural Data Flow Testing",
ACM SIGSOFT Software Engineering Notes,
vol 14 no 8 (Dec 1989), pp 158-167.

[Harrold-91]
Harrold, Mary Jean; Malloy, Brian.
"A Unified Interprocedural Program Representation for a
Maintenance Environment",
Proceedings 1991 IEEE Conference on Software
Maintenance, Sorrento Italy (Oct 1991), pp 138-147.

[Harrold-93]
Harrold, Mary Jean; Malloy, Brian.
"A Unified Interprocedural Program Representation for a
Maintenance Environment",
IEEE Transactions on Software Engineering, vol 19 no 6
(June 1993), pp 584-593.

[Hecht-77]
Hecht, M. S.
Flow Analysis of Computer Programs,
Elsevier, 1977.

[Heitkoetter-90]
Heitkoetter, U.; Helling, B.; Nolte, H.
"Design Metrics and Aids to their Automatic Collection",
Information and Software Technology,
vol 32 no 1 (Jan/Feb 1990), pp 79-87.

[Henry-81]
Henry, S.; Kafura, D.
"Software Structure Metrics Based on Information Flow",
IEEE Transactions on Software Engineering,
vol SE-7 no 5 (Sept 1981), pp 510-518.

[Henry-84]
Henry, S.; Kafura, D.
"The Evaluation of Software Systems' Structure Using Quantitative Software Metrics",
Software Practice and Experience, vol 14 no 6 (1984),
pp 561-573.

[Hoffman-90]
Hoffman, D.
"On Criteria for Module Interfaces",
IEEE Transactions on Software Engineering,
vol 16 no 5 (May 1990), pp 537-542.

[Hoffman-91]
Hoffman, Daniel M.; Strooper, Paul.
"Automated Module Testing in Prolog",
IEEE Transactions on Software Engineering,
vol 17 (Sept 1991), pp 934-943.

[Horwitz-90]
Horwitz, S.; Reps, T.; Binkley, D.
"Interprocedural Slicing Using Dependence Graphs",
ACM Transactions on Programming Languages and Systems,
vol 12 no 1 (Jan 1990), pp 26-60.

[Hunter-86]
Hunter, D.; Kobitzsch, W.
"Measurement Interface Module for the Software
Development Environment",
Electrical Communication, vol 60 no 3-4 (1986),
pp 256-258.

[Ince-85]
Ince, D.
"The Influence of System Design Complexity Research on
the Design of Module Interconnection Languages",
ACM SIGPLAN Notices, vol 20 no 10 (Oct 1985), pp 36-43.

[Ince-89]
Ince, D.C.; Shepperd, M.J.
"An Empirical and Theoretical Analysis of an
Information Flow-Based System Design Metric",
Proceedings 2nd European Conference on Software
Engineering, Coventry UK (Sept 1989), pp 86-99.

[Ince-90a]
Ince, D.
"An Annotated Bibliography of Software Metrics",
ACM SIGPLAN Notices, vol 25 no 8 (Aug 1990), pp 15-23.

[Ince-90b]
Ince, D.; Shepperd, M.
"The Use of Cluster Techniques and System Design Metrics in Software Maintenance",
Proceedings UK IT 1990 Conference,
Southampton UK (March 1990), pp 139-142.

[Jones-91]
Jones, C.
Applied Software Measurement,
McGraw-Hill, 1991.

[Kafura-82]
Kafura, D.; Henry, S.
"Software Quality Metrics Based on Interconnectivity",
The Journal of Systems and Software, vol 2 (1982),
pp 121-131.

[Kaposi-91]
Kaposi, Agnes A.
"Measurement Theory",
Software Engineer's Reference Book,
Butterworth-Heinemann Ltd, 1991.

[Kearney-86]
Kearney, J. K.
"Software Complexity Measurement",
Communications of the ACM, vol 29 no 11 (Nov 1986),
pp 1044-1050.

[Keutgen-81]
Keutgen, H.
A metric for evaluation of the modularization,
Lecture Notes on Computer Science 50, Springer Verlag,
1981.

[Kirchgassner-87]
Kirchgassner, Walter; Persch, Guido; Uhl, Jurgen.
"Structural Analysis of Large Ada Systems",
Ada components: libraries and tools, Proc. Ada-Europe
International Conference, Stockholm 26-28 May 1987.
The Ada Companion Series, Cambridge University Press,
Sven Tafvelin eds.

[Krauskopf-90]
Krauskopf, J.
"Elemental Concerns (Software Design)",
IEEE Potentials, vol 9 no 1 (Feb 1990), pp 13-15.

[Kyu-90]
Kyu Jung Han; Jeong Ah Kim; Kyung Whan Lee.
"A Quality Assessment Criterion of C++ Classes with
Abstract Data Types and Inheritance" (in Korean),
Journal of the Korea Information Science Society,
vol 17 no 5 (Sept 1990), pp 550-559.

[Lakshmanan-91]
Lakshmanan, K. B.; Jayaprakash, S.; Sinha, P. K.
"Properties of Control-Flow Complexity Measures",
IEEE Transactions on Software Engineering,
vol 17 no 12 (Dec 1991), pp 1289-1295.

[Leach-90]
Leach, R.J.
"Software Metrics and Software Maintenance",
Journal of Sofware Maintenance: Research and Practice,
vol 2 no 2 (June 1990), pp 113-142.

[Leavens-91]
Leavens, Gary T.
"Modular Specification and Verification of Object-Oriented Programs",
IEEE Software, vol 8 (July 1991), pp 72-80.

[Lee-87]
Lee, Tony T.
"An Information-Theoric Analysis of Relational Databases I. Data Dependencies and Information Metric",
IEEE Transactions on Software Engineering,
vol 13 no 10 (1987), pp 1049-1061.

[Lee-91]
Lee, Edward Ashford.
"Consistency in Dataflow Graphs",
IEEE Transactions on Parallel and Distributed Systems,
vol 2 (April 1991), pp 223-235.

[Liu-90]
Liu, Sying-Syang; Wilde, Norman.
"Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery",
Proceedings 1990 IEEE Conference on Software Maintenance, San Diego Calif. (Nov 1990), pp 266-271.

[Lohse-84]
Lohse, J. B.; Zweben, S. H.
"Experimental Evaluation of Software Design Principles: an Investigation into the Effect of Module Coupling on System and Modifiability",
Journal of Systems and Software, vol 4 (1984),
pp 301-308.

[Lowe-91]
Lowe, M.; Ehrig, H.; Fey, W.; Jacobs, D.
"On the Relationship Between Algebraic Module Specifications and Program Modules",
Proceedings International Joint Conference on Theory and Practice of Software Development Vol 2: Advances in Distributed Computing (ADC) and Colloquium on Combining Paradigms for Software Development (CCPSD),
Brighton UK (April 1991).

[Luttger-90]
Luttger, J.; Pauthner, G.; Schulengerg, H.
"Context Data Management System",
Electrical Communication, vol 64 no 4 (1990),
pp 341-347.

[Lyle-88]
Lyle, J. R.; Gallagher, K. B.
"Using Program Decomposition to Guide Modification",
Proceedings 1988 IEEE Conference on Software Maintenance, Phoenix Arizona (Oct 1988), pp 265-269.

[Maarek-88]
Maarek, Y.S.
"On the Use of Cluster Analysis for Assisting Maintenance of Large Software Systems",
Proceedings 3rd Israel Conference on Computer Systems and Software Engineering, Tel-Aviv Israel (June 1988),
pp 178-186.

[MacLennan-87]
MacLennan, Bruce J.
Principles of Programming Languages
Design, Evaluation, and Implementation,
Holt, Rinehart and Winston, New York, 1987.

[McAuliffe-88]
McAuliffe, Daniel.
"Measuring Program Complexity",
Computer, vol 21 (June 1988), pp 97-98.

[McCabe-89]
McCabe, Thomas J.; Butler, Charles W.
"Design Complexity Measurement and Testing",
Communications of the ACM, vol 32 no 12 (Dec 1989),
pp 1415-1425.

[Mennert-91]
Mennert, A.
Measuring Control Flow Complexity for Software Development,
Technical Report, Siemens, Princeton, New Jersey, 1991.

[Mitchell-88]
Mitchell, R. J.
"Applying the Principle of Separation of Concerns in Software Development",
Proceedings 15th IFAC/IFIP Workshop on Real Time Programming, Valencia Spain (May 1988), pp 21-27.

[Mizuno-89]
Mizuno, M.
"An Iterative Method for Secure Inter-Procedural Information Flow Control",
Proceedings 13th Annual International Computer Software and Applications Conference, Orlando Florida (Sept 1989), pp 286-291.

[Mohanty-81]
Mohanty, S. B.
"Entropy Metrics for Software Design Evaluation",
The Journal of Systems and Software, vol 2 (1981),
pp 39-46.

[Moser-90]
Moser, Louise E.
"Data Dependency Graphs for Ada Programs",
IEEE Transactions on Software Engineering,
vol 16 no 5 (May 1990), pp 498-509.

[Muhanna-91]
Muhanna, Waleed A.
"Composite Programs: Hierarchical Construction,
Circularity, and Deadlocks",
IEEE Transactions on Software Engineering,
vol 17 (April 1991), pp 320-333.

[Muller-89]
Muller, John H.; Spinrad, Jeremy.
"Incremental Modular Decomposition",
Journal of the Association for Computing Machinery,
vol 36 no 1 (Jan 1989), pp 1-19.

[Muller-90]
Muller, Hausi A.; Uhl, James S.
"Composing Subsystem Structures Using (K,2)-Partite
Graphs",
Proceedings 1990 IEEE Conference on Software
Maintenance, San Diego Calif. (Nov 1990), pp 12-19.

[Murtagh-84]
Murtagh, Thomas P.
"A Less Dynamic Memory Allocation Scheme for Algol-like Languages",
Proceedings 11th Annual ACM Symposium on Principles of Programming Languages, Salt Lake City Utah (Jan 1984), pp 283-289.

[Murtagh-91]
Murtagh, Thomas P.
"An Improved Storage Management Scheme for Block Structured Languages",
ACM Transactions on Programming Languages and Systems, vol 13 no 3 (July 1991), pp 372-398.

[Myers-75]
Myers, Glenford J.
Reliable Software Through Composite Design,
Petrocelli/Charter, 1975.

[Myers-77]
Myers, Glenford J.
"An Extension to the Cyclomatic Measure of Program Complexity",
ACM SIGPLAN Notices, vol 12 no 10 (Oct 1977), pp 61-64.

[Myers-78]
Myers, Glenford J.
Composite / Structured Design,
Van Nostrand Reinhold, 1978.

[Nakagawa-89]
Nakagawa, A.T.; Futatsugi, K.
"Stepwise Refinement Process with Modularity:
an Algebraic Approach",
Proceedings 11th International Conference on Software
Engineering, Pittsburgh PA USA (May 1989), pp 166-177.

[Nani-90]
Nani, G.
"Comparing the Effectiveness of Decomposition Tools",
Advances in Modelling & Simulation, vol 19 no 3 (1990),
pp 13-36.

[Nass-91]
Nass, Richard.
"Maintain and Reengineer Existing C Programs",
Electronic Design, vol 39 (April 1991), p 154.

[Navlakha-87]
Navlakha, J. K.
"A Survey of System Complexity Metrics",
The Computer Journal, vol 30 no 3 (June 1987),
pp 233-238.

[Nejmeh-88]
Nejmeh, Brian A.
"NPATH: a Measure of Execution Path Complexity and its
Applications",
Communications of the ACM, vol 31 (Feb 1988),
pp 188-200.

[Nielsen-86]
Nielsen, Kjell W.
"Task Coupling and Cohesion in Ada",
ACM SIGAda Ada Letters, vol 6 no 4 (July, August 1986),
pp 44-52.

[Olson-90]
Olson, J.W.; Doran, M.V.; Longenecker, H.E., Jr.
"The Establishment and Application of a Metric for
Graphical Design Language",
Proceedings 18th ACM Annual Conference on Computer
Science, Washington DC (Feb 1990), p 407.

[Ott-89]
Ott, L.M.; Thuss, J.J.
"The Relationship Between Slices and Module Cohesion",
Proceedings 11th International Conference on Software
Engineering, Pittsburgh PA (May 1989), pp 198-204.

[Oval-89]
Oval, Francois; Meuleau, Jean-Charles
"Mesure de l'Independance d'un Graphe d'Appel dans une
Application",
Proceedings 2nd International Workshop on Software
Engineering and its Applications, Toulouse France
(1989), pp 963-969.

[Overstreet-88]
Overstreet, C. Michael; Chen, Ji; Byrum, Frank.
"Program Maintenance by Safe Transformations",
Proceedings 1988 IEEE Conference on Software
Maintenance, Phoenix Arizona (Oct 1988), pp 118-123.

[Oviedo-80]
Oviedo, E. I.
"Control Flow, Data Flow, and Program Complexity",
Proceedings IEEE Computer Society's 4th International
Computer Software & Applications Conference (COMPSAC),
Chicago USA (Oct 1980), pp 146-152.

[Parisi-Presicce-90]
Parisi-Presicce, F.
"A Rule-Based Approach to Modular System Design",
Proceedings 12th International Conference on Software-
Engineering, Nice France (March 1990), pp 202-211.

[Parisi-Presicce-91]
Parisi-Presicce, F.
"Foundations of Rule-Based Design of Modular Systems",
Theoretical Computer Science, vol 83 no 1 (June 1991),
pp 131-155.

[Parnas-71]
Parnas, D. L.
"Information Distribution Aspects of Design
Methodology",
Proceedings IFIP Congress, Ljubljana Yougoslavia
(1971), pp 339-344.

[Parnas-72a]
Parnas, D. L.
"A Technique for Software Module Specification with
Examples",
Communications of the ACM, vol 15 (May 1972),
pp 330-336.

[Parnas-72b]
Parnas, D. L.
"On the Criteria to be Used in Decomposing Systems into Modules",
Communications of the ACM, vol 15 (Dec 1972),
pp 1053-1058.

[Parnas-85]
Parnas, D. L.; Clements, P. C.; Weiss, D. M.
"The Modular Structure of Complex Systems",
IEEE Transactions on Software Engineering,
vol SE-11 no 3 (March 1985), pp 259-266.

[Paulson-92]
Paulson, Dan; Wand, Yair.
"An Automated Approach to Information Systems Decomposition",
IEEE Transactions on Software Engineering,
vol 18 no 3 (March 1992), pp 174-189.

[Power-90]
Power, L. R.
"Post-Facto Integration Technology:
New Discipline for an Old Practice",
Proceedings 1st International Conference on Systems Integration, Morristown NJ (April 1990), pp 4-13.

[Prather-84]
Prather, Ronald E.
"An Axiomatic Theory of Software Complexity Measure",
The Computer Journal, vol 27 no 4 (1984), pp 340-347.

[Pugh-92]
Pugh, William.
"Definition of Dependance Distance",
ACM Letters on Programming Languages and Systems,
vol 1 no 3 (Sept 1992), pp 261-265.

[Purtilo-90]
Purtilo, J. M.; Atlee, J. M.
"Improving Module Reuse by Interface Adaptation",
Proceedings IEEE 1990 International Conference on
Computer Languages, New Orleans LA (March 1990),
pp 208-217.

[Ramamoorthy-86]
Ramamoorthy, C. V.; Garg, V.; Prakash, A.
"Programming in the Large",
IEEE Transactions on Software Engineering,
vol 12 no 7 (1986), pp 769-783.

[Reynolds-84]
Reynolds, R. G.
"Metrics to Measure the Complexity of Partial
Programs",
The Journal of Systems and Software, no 4 (1984),
pp 75-91.

[Reynolds-90]
Reynolds, R. G.; Maletic, J. I.
"An Introduction to Refinement Metrics: Assessing a
Programming
Language's Support of the Stepwise Refinement Process",
Proceedings 18th ACM Annual Conference on Computer
Science, Washington DC (Feb 1990), pp 82-88.

[Richardson-89]
Richardson, S.; Ganapathi, M.
"Interprocedural Analysis vs Procedure Integration",
Information Processing Letters, vol 32 no 3 (Aug 1989),
pp 137-142.

[Rising-92]
Rising, Linda; Caliss, Frank W.
"Problems with Determining Package Cohesion and
Coupling",
Software Practice and Experience,
vol 22 no 7 (July 1992), pp 553-571.

[Roberts-79]
Roberts, Fred S.
Measurement Theory,
Encyclopedia of Mathematics and its Applications Vol 7,
Addison-Wesley, 1979.

[Robillard-89]
Robillard, Pierre N.; Boloix, Germinal.
"The Interconnectivity Metrics: A New Metric Showing
How a Program is Organized",
The Journal of Systems and Software, vol 10 (1989),
pp 29-30.

[Robillard-91]
Robillard, Pierre N.; Coupal, Daniel;
Coallier, François.
"Profiling Software Through the Use of Metrics",
Software Practice and Experience,
vol 21 no 5 (May 1991), pp 507-518.

[Rombach-84]
Rombach, H. D.
Quantitative evaluation of software quality characteristics on the base of structurally measures (in German),
Thèse, Universitat Kaiserslautern, 1984.

[Ross-86]
Ross, Donald L.
"Classifying Ada Packages",
ACM SIGAda Ada Letters, vol 6 no 4 (July/August 1986), pp 53-65.

[Ryder-79]
Ryder, Barbara G.
"Constructing the Call Graph of a Program",
IEEE Transactions on Software Engineering, vol SE-5 no 3 (May 1979), pp 216-230.

[Ryder-90]
Ryder, Barbara Gershon; Landi, William; Pande, Hemant.
"Profiling an Incremental Data Flow Analysis Algorithm",
IEEE Transactions on Software Engineering, vol 16 no 2 (Feb 1990), pp 129-140.

[Sampson-87]
Sampson, W. B.; Nevill, D. G.; Dugard, P. I.
"Predictive Software Metrics Based on a Formal Specification",
Information and Software Technology, vol 29 (June 1987), pp 242-248.

[Schach-90]
Schach, Stephen R.
SOFTWARE ENGINEERING,
Aksen Associates, 1990.

[Schneidewind-91]
Schneidewind, Norman F.
"Setting Maintenance Quality Objectives and Prioritizing Maintenance Work by Using Quality Metrics",
Proceedings 1991 IEEE Conference on Software Maintenance, Sorrento Italy (Oct 1991), pp 240-249.

[Seidewitz-87]
Seidewitz, Ed; Stark, Mike.
"Towards a General Object-Oriented Software Development Methodology",
Ada Letters, vol 7 no 4 (1987), pp 54-67.

[Selby-88]
Selby, Richard W.; Basili, Victor R.
"Error Localization During Maintenance: Generating Hierarchical System Descriptions from the Source Code Alone",
Proceedings 1988 IEEE Conference on Software Maintenance, Phoenix Arizona (Oct 1988), pp 192-197.

[Sellers-92]
Sellers, B. Henderson.
"Modularization and McCabe's Cyclomatic Complexity",
Communications of the ACM, vol 35 no 12 (Dec 1992), pp 17-19.

[Shepperd-90]
Shepperd, M.; Ince, D.
"Multi-Dimensional Modelling and Measurement of Software Designs",
Proceedings 18th ACM Annual Conference on Computer Science, Washington DC (Feb 1990), pp 76-81.

[Shumate-88]
Shumate, Ken; Nielsen, Kjell.
"A Taxonomy of Ada Packages",
ACM Ada Letters, vol 8 no 2 (1988), pp 55-76.

[Sommerville-89]
Sommerville, Ian.
SOFTWARE ENGINEERING,
Addison-Wesley, 1989.

[Stevens-74]
Stevens, W. P.; Myers, G. J.; Constantine, L. L.
"Structured Design",
IBM Systems Journal, vol 13 no 2 (1974), pp 115-139.

[Stubbs-84]
Stubbs, Michael.
"An Examination of the Resolution of Structure Clashes by Structure Inversion",
The Computer Journal, vol 27 (Nov 1984), pp 354-361.

[Tai-84]
Tai, K.
"A Program Complexity Metric Based on Data Flow Information Control Graphs",
Proceedings 7th International Conference on Software Engineering, Orlando Florida (March 1984), pp 239-248.

[Teufel-91]
Teufel, Bernd.
Organization of Programming Languages,
Springer-Verlag, Vienne, 1991.

[Torres-91]
Torres, W. R.; Samadzadeh, M.
"Software Reuse and Information Theory Based Metrics",
Proceedings 1991 Symposium on Applied Computing,
Kansas City April 1991, IEEE Computer Society Press,
pp 437-446.

[Troy-81]
Troy, D. A.; Zweben, S. H.
"Measuring the Quality of Structured Designs",
The Journal of Systems and Software, vol 2 (June 1981),
pp 113-120.

[Turner-80]
Turner, J.
"The Structure of Modular Programs",
Communications of the ACM, vol 23 no 5 (May 1980),
pp 272-277.

[VanEmden-70]
Van Emden, M. H.
"Hierarchical Decomposition of Complexity",
Machine Intelligence, vol 5 (1970), pp 361-380.

[VanVerth-87]
Van Verth, P. B.
"A Program Complexity Model that Includes Procedures",
Buffalo, 1987.

[Vanek-89]
Vanek, Leonard I.; Culp, Mark N.
"Static Analysis of Program Source Code using EDSA",
Proceedings 1989 IEEE Conference on Software
Maintenance, Miami Florida (Oct 1989), pp 192-199.

[Watanabe-60]
Watanabe, S.
"Information Theoretical Analysis of Multivariate
Correlation",
IBM Journal, vol 4 (1960), pp 66-82.

[Watson-87]
Watson, S. E.
"Ada Modules",
ACM SIGAda Ada Letters, vol 7 no 4 (July, August 1987),
pp 79-84.

[Weiss-92]
Weiss, Michael.
"The Transitive Closure of Control Dependence",
ACM Letters on Programming Languages and Systems,
vol 1 no 2 (June 1992), pp 178-190.

[Weyuker-88]
Weyuker, E. J.
"Evaluating software complexity measures",
IEEE Transactions on Software Engineering,
vol 14 (Sept 1988), pp 1357-1365.

[Whitworth-80]
Whitworth, Mark H.; Szulewski, Paul A.
"The Measurement of Control and Data Flow Complexity in Software Designs",
Proceedings IEEE Computer Society's 4th International Computer Software and Applications Conference (COMPSAC), Chicago USA (1980), pp 735-743.

[Wilde-89]
Wilde, Norman; Huitt, Ross; Huitt, Scott.
"Dependency Analysis Tools: Reusable Components for Software Maintenance",
Proceedings 1989 IEEE Conference on Software Maintenance, Miami Florida (Oct 1989), pp 126-131.

[Woodfield-80]
Woodfield, S. N.
Enhanced Effort Estimation by Extending Basic Programming Model to Include Modularity Factors,
Thèse, Dept of Computer Sc. Purdue University,
Dec 1980.

[Woodfield-81a]
Woodfield, S. N.; Dunsmore, H. E.; Shen, V. Y.
"The Effect of Modularization and Comments on Program Comprehension",
Proceedings 5th International Conference on Software Engineering (March 1981), pp 215-222.

[Woodfield-81b]
Woodfield, S. N.; Shen, V. Y.; Dunsmore, H. E.
"A Study of Several Metrics for Programming Effort",
The Journal of Systems and Software,
vol 2 no 2 (Dec 1981), pp 97-103.

[Woodward-79]
Woodward, M. R.; Hennel, M. A.; Hedley, D.
"A Measure of Control Flow Complexity in Program Text",
IEEE Transaction on Software Engineering,
vol SE-5 no 1 (Jan 1979), pp 45-50.

[Yadav-90]
Yadav, S. B.
"Control and Definition Modularization: an Improved
Software Design Technique for Organizing Programs",
IEEE Transactions on Software Engineering,
vol 16 no 1 (Jan 1990), pp 92-99.

[Yau-80]
Yau, S. S.; Grabow, P. C.
"A Model for Representing the Control Flow and Data
Flow of Program Modules",
Proceedings IEEE Computer Society's 4th International
Computer Software & Applications Conference (COMPSAC),
Chicago USA (Oct 1980), pp 153-160.

[Yau-91]
Yau, S. S.; Wiharja, I.
"An Approach to Module Distribution for the Design of
Embedded Distributed Software Systems",
Information Sciences, vol 56 no 1-3 (Aug 1991),
pp 1-22.

[Yaung-92]
Yaung, Alan T.; Raz, Tzvi.
"Linkage Analysis of Processes",
Software Practice and Experience,
vol 22 no 10 (Oct 1992),
pp 849-862.

[Yourdon-79]
Yourdon, Edward; Constantine, Larry L.
STRUCTURED DESIGN,
Prentice-Hall, 1979.

[Zuse-91]
Zuse, Horst.
SOFTWARE COMPLEXITY MEASURES AND METHODS,
Walter de Gruyter Inc, 1991.