Building a Framework for Coupling Measurement Systems

by

Tariq S. Al-Zubaidi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

December, 1998

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

BUILDING A FRAMEWORK FOR COUPLING MEASUREMENT SYSTEMS

BY

TARIQ S. ALZUBAIDI

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

DECEMBER, 1998

UMI Number: 1395611

UMI Microform 1395611
Copyright 1999, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized
copying under Title 17, United States Code.

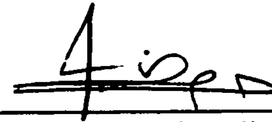
UMI
300 North Zeeb Road
Ann Arbor, MI 48103

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **TARIQ SALIM ALZUBAIDI** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

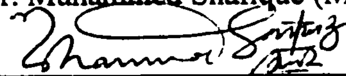
Thesis Committee



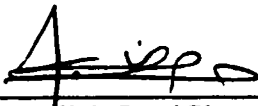
Dr. Jarallah S. AlGhamdi (Chairman)

Dr. Muslim Bozyigit (Member)

Dr. Muhammed Shafique (Member)

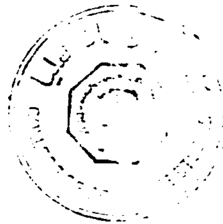


Dr. Muhammed Sarfraz (Member)



Dr. Jarallah S. AlGhamdi
Chairman, Department of Information and Computer Science

Dr. Abdullah M. Al-Shehri
Dean of Graduate Studies



12/5/99
Date

Acknowledgement

I would like to acknowledge King Fahd University of Petroleum and Minerals for all the support extended during this research.

I would like to extend my appreciation to my thesis committee chairman, Dr. Jarallah S. Alghamdi for his continuous advice, guidance and cooperation. I also thank the other members of my thesis committee Dr. Muhammed Shafique, Dr. Muhammed Sarfraz and Dr. Muslim Bozyigit for their suggestions and cooperation.

Thanks to all colleagues and friends for their suggestions and encouragement during the preparation of this thesis.

I would like to express my sincere thanks and gratitude to my mother, my wife and my children for their support, patience and sacrifices.

Contents

ACKNOWLEDGEMENT II

CONTENTS III

LIST OF TABLES VI

LIST OF FIGURES VIII

ABSTRACT IX

ABSTRACT (ARABIC) X

1 INTRODUCTION 1

1.1 SOFTWARE METRICS 2
1.1.1 Quantity Metrics 2
1.1.2 Quality Metrics 3
1.1.2.1 Functionality 3
1.1.2.2 Portability 3
1.1.2.3 Reliability 4
1.1.2.4 Security 4
1.1.2.5 Efficiency 4
1.1.2.6 Maintainability 5
1.1.3 Complexity Metrics 5
1.1.4 Coupling 6
1.2 PROBLEM DEFINITION 7
1.3 THESIS ORGANIZATION 8

2 COUPLING METRICS - BACKGROUND 9

2.1 FENTON AND MELTON METHOD 9
2.1.1 Coupling Attributes 12
2.2 OFFUT, HARROLD AND KOTLE METHOD 12
2.2.1 Coupling Attributes 14
2.3 DHAMA'S METHOD 15
2.3.1 Coupling Attributes 17
2.4 COVER COEFFICIENT CONCEPT 17
2.4.1 Al-Nasser Weighing Scheme 21
2.4.2 Coupling Attributes 25
2.5 MOTIVATION OF THE WORK 26

3 THE FRAMEWORK OF COUPLIMG MEASUREMENT SYSTEM DESIGN 28

3.1 SYSTEM STAGES 29
3.1.1 Stage One 29
3.1.2 Stage Two 30
3.1.3 Stage Three 30
3.2 DEVELOPMENT PHASES OF THE ABSTRACT FORMAT 31
3.3 BENEFITS OF THE FCMS 41
4 SYSTEM IMPLEMENTATION 43
4.1 STAGE ONE IMPLEMENTATION 43
4.1.1 System Attributes Extraction 45
4.1.1.1 Type declaration 45
4.1.1.2 Functions declaration 47
4.1.1.3 Parameters declaration 47
4.1.1.4 Global variables declaration 48
4.1.1.5 Local variables declaration 49
4.1.1.6 Function Calls 50
4.1.1.7 Passed Arguments 51
4.1.1.8 Variables that appear at LHS of assignment 52
4.1.1.9 Variables that appear at RHS of assignment 52
4.1.1.10 Variables that appear in logical expression 52
4.1.1.11 Variables passed as arguments in function calls 53
4.2 STAGE TWO IMPLEMENTATION 54
4.2.1 Populating the system attributes into the database 54
4.2.2 Creating System calls and their formal parameters 54
4.3 STAGE THREE IMPLEMENTATION 55
4.3.1 Dhama's method 55
4.3.2 Cover-Coefficient method 57
4.3.2.1 Main body of the algorithm 59
4.3.3 Resolving Aliases 61
4.3.3.1 Uncovering aliases algorithm 63
4.4 IMPLEMENTATION OF MATRICES 66
5 STUDY CASES 68
5.1 COVER COEFFICIENT METHOD 68
5.1.1 Case 1 68
5.1.2 Case 2 69
5.2 DHAMA'S METHOD 76
5.2.1 Case 1 76
5.2.2 Case 2 78
6 CONCLUSION 81
6.1 CONTRIBUTION 81
6.2 FUTURE WORKS 83

APPENDIX A **84**

APPENDIX B **89**

APPENDIX C **117**

APPENDIX D **150**

APPENDIX E **157**

APPENDIX F **159**

APPENDIX G **162**

REFERENCES **171**

List of Tables

TABLE 1. FENTON AND MELTON MODIFIED DEFINITION FOR MYERS COUPLING LEVELS. 10
TABLE 2. OFFUT, HARROLD AND KOTLE MODIFIED DEFINITION OF MYERS COUPLING LEVEL. 13
TABLE 3. DEFINITION MATRIX OF THE EXAMPLE 2.2. 20
TABLE 4. COUPLING MATRIX OF THE EXAMPLE 2.2. 21
TABLE 5 AL-NASSER PROPOSED WEIGHT OF COUPLING 23
TABLE 6. SCHEMA DEFINITION FOR TABLE SYSTEM. 35
TABLE 7. SCHEMA DEFINITION FOR TABLE MODULE. 36
TABLE 8. SCHEMA DEFINITION FOR TABLE FUNCTION. 36
TABLE 9. SCHEMA DEFINITION FOR TABLE PARAMETER. 37
TABLE 10. SCHEMA DEFINITION FOR TABLE GLOBAL_VAR. 37
TABLE 11. SCHEMA DEFINITION FOR TABLE LOCAL_VAR. 38
TABLE 12. SCHEMA DEFINITION FOR TABLE DATA-TYPE. 38
TABLE 13. SCHEMA DEFINITION FOR TABLE TYPE-FIELD. 39
TABLE 14. SCHEMA DEFINITION FOR FUN-CALL. 39
TABLE 15. SCHEMA DEFINITION FOR PASSED-ARGUMENT. 40
TABLE 16. SCHEMA DEFINITION FOR TABLE SYS-FUN. 40
TABLE 17. SCHEMA DEFINITION FOR TABLE SYS-PARAMETER. 41
TABLE 18. LIST OF RESOLVED PARAMETER ALIASES. 66
TABLE 19. CC DEFINITION MATRIX FOR PROGRAM SORT 1 70
TABLE 20. CC COUPLING MATRIX FOR PROGRAM SORT 1 70
TABLE 21. CC SYSTEM AND AVERAGE COUPLING OF PROGRAM SORT 1. 70
TABLE 22. CC DEFINITION MATRIX OF SORT 11. 71
TABLE 23. CC COUPLING MATRIX OF SORT 11. 71
TABLE 24. CC SYSTEM AND AVERAGE COUPLING OF SORT11. 71
TABLE 25. CC DEFINITION MATRIX OF PROGRAM SORT 1. (MODIFIED ALOGRITHM). 71
TABLE 26. CC COUPLING MATRIX OF PROGRAM SORT 1. (MODIFIED ALGORITHM). 72
TABLE 27. CC SYSTEM AND AVERAGE COUPLING OF PROGRAM SORT 1 (MODIFIED ALGORITHM). 72
TABLE 28. CC DEFINITION MATRIX OF SORT 2. 72
TABLE 29. CC COUPLING MATRIX OF SORT 2. 72
TABLE 30. CC SYSTEM AND AVERAGE COUPLING OF SORT 2. 72
TABLE 31. CC DEFINITION MATRIX OF SORT 21. 73
TABLE 32. CC COUPLING MATRIX OF SORT 21. 73
TABLE 33. CC SYSTEM AND AVERAGE COUPLING OF SORT 21 73

TABLE 34. CC DEFINITION MATRIX OF PROGRAM SORT2. (MODIFIED ALGORITHM). 73
TABLE 35. CC COUPLING MATRIX OF PROGRAM SORT 2.(MODIFIED ALGORITHM). 74
TABLE 36. CC SYSTEM AND AVERAGE COUPLING OF SORT 2. (MODIFIED ALGORITHM). 74
TABLE 37. CC DEFINITION MATRIX OF SORT 3. 74
TABLE 38. CC COUPLING MATRIX OF SORT 3. 74
TABLE 39. CC SYSTEM AND AVERAGE COUPLING OF SORT 3. 74
TABLE 40. CC DEFINITION MATRIX OF SORT 31. 75
TABLE 41. CC COUPLING MATRIX OF SORT 31. 75
TABLE 42. CC SYSTEM AND AVERAGE COUPLING OF SORT 31 75
TABLE 43. CC DEFINITION MATRIX OF PROGRAM SORT 3 (MODIFIED ALGORITHM). 75
TABLE 44. CC COUPLING MATRIX OF PROGRAM SORT 3 (MODIFIED ALGORITHM). 76
TABLE 45. CC SYSTEM AND AVERAGE COUPLING OF PROGRAM SORT 3 (MODIFIED ALGORITHM). 76
TABLE 46. DHAMA'S SYSTEM COUPLING OF GCMS. 78
TABLE 47. DHAMA'S SYSTEM COUPLING OF SORT 1. 78
TABLE 48. DHAMA'S SYSTEM COUPLING FOR SORT 11. 78
TABLE 49. DHAMA'S SYSTEM COUPLING FOR SORT 2. 79
TABLE 50. DHAMA'S SYSTEM COUPLING FOR SORT 21. 79
TABLE 51. DHAMA'S SYSTEM COUPLING FOR SORT 3. 79
TABLE 52. DHAMAS SYSTEM COUPLING FOR SORT 31. 80

List of Figures

FIGURE 1. EXAMPLE OF FENTON AND MELTON METHOD. 11
FIGURE 2. THE THREE SYSTEM STAGES 29
FIGURE 3. E-R DIAGRAM OF THE SYSTEM 35
FIGURE 4. TYPE NAME ALIASES. 46
FIGURE 5. NESTED COMPLEX DATA STRUCTURES. 47
FIGURE 6. OLD STYLE PARAMETERS DECLARATION. 48
FIGURE 7. ANSI STANDARD PARAMETERS DECLARATION. 48
FIGURE 8. USER-DEFINED DATA TYPE WITH NO TYPE-TAG 49
FIGURE 9. FUNCTIONS CALLS SEQUENCE. 50
FIGURE 10. ALGORITHM THAT CALCULTES DHAMA'S COUPLING METRICS..... 57
FIGURE 11. ALGORITHM THAT CALCULATES CC COUPLING METRICS. 61
FIGURE 12. PROCEDURE THAT UPDATES ALL ALIASES OF CC MATRIX 61
FIGURE 13. EXAMPLE OF PARAMETERS ALIASES. 62
FIGURE 14. ALGORITHM USED TO UNCOVERING PARAMETER ALIASES. 65
FIGURE 15. PROCEDURE USED TO UNCOVER FIRST LEVEL ALIASES. 65
FIGURE 16. MATRIX REPRESENTATION IN RELATIONAL TEMP TABLES 67

Abstract

Name: Tariq Salim Alzubaidi

Title: Building a Framework for Coupling Measurement Systems

Degree: Master of Science

Major Field: Information & Computer Science

Date of Degree: December 1998

Software metrics play a major role in determining the quality and the maintainability of software systems. Coupling metrics are among the software metrics that measure the interconnections of software components. There is a need to have a system acting as standard platform for different coupling metrics. Such system should be capable to collect, store and analyze necessary coupling data automatically.
A general method of measuring coupling metrics is presented. The method is based on transforming the program constructs of imperative high level languages into a higher abstract format. The abstract format is language and environment independent. It captures the necessary attributes needed to calculate the coupling metrics while hiding unnecessary details of the language. The abstract format is mapped into relational model and maintained by a relational database management system(RDBMS).
A framework for coupling measurement system (FCMS) based on the above method has been developed. It is a three stages system. The first stage involves parsing a C language source code to extract the coupling attributes. Second stage involves in uploading these attributes into the RDBMS. The final stage involves developing modules using a 4GL language to calculate the coupling metrics using Dhama's and Cover-Coefficient metric methods. The modules access the Database to retrieve their input parameters. The FCMS was experimented using a set of production programs and their coupling metrics were computed.

Keywords: Softwar Metrics, Coupling Metrics, Coupling Data, Imperative High Level Language, Abstract Format, Relational Model, Relational Database Management System, Coupling Attributes, 4GL.

King Fahd University of Petroleum and Minerals, Dhahran.

December 1998

خلاصة الرسـالة

أسـم الطالـب : طارق سالم الزبيدي
عنوان الدراسة : بناء إطار لأنظمة طرق قياس الارتباط
التخصـص :معلومات وعلوم الحاسب الآلي
تاريخ الشـهادة : ديسمبر ١٩٩٨

تلعب طرق قياس البرامج دوراً مهمـاً في تحديد نوعية وطرق صيانة برامج الحاسب الآلـي. وتعتبر طرق قياس الارتباط إحدى أنواع طرق مقـاييس الـبرامج التـي تقيـس مكونـات الموصلات المتداخلة لمكونات برامج الحاسب. هناك حاجة ماسة لنظام يمثل قاعدة ثابتة لكافة طرق مقاييس الارتباط ويجب أن يكون هذا النظام قادراً على جمع وتخزين وتحليل مقاييس أنظمة الربط تلقـائيـاً.

ولقد تم في هذه الرسالة عرض طريقة جديدة تعتمد على تحويل لبنـات لغـات البرمجـة التركيبية العالية إلى مستوى تجريدي أعلى . هذا المستوى التجريدي غير مرتبط بـأي لغـة برمجة عالية أو أي نظام تشغيلي ويحتوي على المكونات الأساسية اللازمة للقيام بحسـابات الترابط ويخفي المكونات الغير ضرورية بحسابات الترابط. لـقد تـم تحويـل المسـتوى التجريدي إلى نموذج نسبـي ليتم بعد ذلك حفظه وتخزينه في قاعدة بيانات نسبية.

لقد تم في هذه الرسالة تطوير إطار لقياس ارتباط مكونات البرامج يعتمـد علـى المسـتوى التجريدي المذكور أعلاه. ويتكون هذا الإطار من ثلاث مراحل رئيسيه، تقوم المرحلة الأولـى بقراءة أي برنامج مكتوب بلغة البرمجة سي لأستخرج المكونات الأساسية للقيـام بحسـابات الترابط. أما المرحلة الثانية فتعتمد على تحميل المكونات المستخرجة في المرحلة الأولى إلـى قاعدة بيانات نسبية. أما المرحلة الثالثة فهي عبارة عن تطوير برامج مكتوبة بلغات البرمجـة ذات الجيل الرابع للقيام بحسـابات الترابط المختلفة وبالأخص طريقة دامـا وطريقة تغطية المعامل. تعتمد البرامج المطورة في المرحلة الثالثة على استخراج الدالات اللازمـة لقيـاس عملية الترابط من قاعدة البيانات النسبية.و لقد تم تجربة الإطار المطور فـي هـذه الرسـالة بتطبيقه على برامج كبيرة وبذلك تم حسـاب ترابط هذه البرامج.

مفاتيح :طرق قياس البرامج، طرق قياس الترابط، معلومات الترابط، لغات البرمجة التركيبية العالية، المستوى التجريدي، نموذج نسبي، نظام قاعدة بيانـات نسـبية، مكونـات حسـابات الترابط، لغات البرمجة ذات الجيل الرابع.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن
الظهران، المملكة العربية السعودية

ديسمبر ١٩٩٨م

Chapter 1

Introduction

Software becomes increasingly important in all fields. It grows very fast and large so that, software development, maintenance and quality assurance tasks become difficult and expensive process. There is a growing need to assess development, quality and re-engineering tasks. Any software product assessments will require a metric scheme to *collect, store* and *analyze* software [Bach94].

In this thesis, an overview of different kinds of coupling metrics are presented. Four coupling metrics are discussed in detailed. A framework for coupling measurement system is presented (FCMS) and details of its implementation are explained. Finally the outputs produced by the system for some production programs are analyzed. The FCMS is capable of parsing any C source code and producing the system description of the parsed source code in flat files format. These files are in turn uploaded into a relational database. The relationships of the tables in the database represent the relational model of the structured programming constructs needed to accomplish the coupling metrics calculation.

The FCMS is able to provide a general method to automatically collect, store and analyze system coupling.

1.1 Software Metrics

The term software metrics means simply measurement applied to software, it is used to assess the quality, effort and re-engineering tasks required. However, measurement should objectively describe some attributes of an object under consideration. There should be a well-defined procedure for arriving at a particular numerical value capturing the named attributes, therefore subjective rating of an object attributes is considered as improper [Gill92].

There are three basic types of software metrics to calculate maintenance, efforts, quality assurance and re-engineering tasks [Sned95]:

1.1.1 Quantity Metrics

Quantity metrics are absolute measures or counts of software characteristics [Sned95].

They are indicators of size or volume of a given software mass.

Examples of such metrics are:

- The number of keys in relational table.
- The number of lines of source code i.e. LOC.
- The number of fields in a map.
- The number of work flow procedures.
- The number of different data types in a data structure.
- The number of statements in a program.

1.1.2 Quality Metrics

Quality metrics are the most difficult to define and to assess [Sned95]. It is an oversimplification to claim that quality is the absence of complexity.

The standard quality characteristics are defined in ISO-9126 [Bach94]. Some of these characteristics can be measured statically as discussed in the rest of this subsection.

1.1.2.1 Functionality

Functionality is defined as 'a set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs'. Hence can be measured as functions coded relative to functions specified [Bach94].

1.1.2.2 Portability

Portability is defined as 'a set of attributes that bear on the ability of the software to be transferred from one environment to another' [Bach94]. Hence can be measured as ratio of standard statements to the total number of statements, considering the fact that in case of migration, standard statements do not have to be altered.

The following characteristics can not be measured statically but by dynamic analysis and extensive field tests [Sned95]; such characteristics are:

1.1.2.3 Reliability

Reliability consists of two sub-characteristics:

- Maturity:

  Maturity is defined as 'the frequency of failure by faults in the software' [Bach94]. Hence, can be measured by mean time to failures MTTF.

- Fault tolerance and recoverability:

  Fault tolerance and recoverability are defined 'as a set of attributes of software that bear on their abilities to maintain a specified level of performance in cases of software faults or infringement of its specified interface' [Bach94].

Such characteristics are difficult to assess, since each case is assessed individually based on specific domain.

1.1.2.4 Security

Security is defined as 'a set of attributes that bear on its ability to prevent unauthorized access, whether accidental or deliberate, to programs and data' [Bach94]. This can be measured by the counts of successful unauthorized access to the total counts of attempted unauthorized access.

1.1.2.5 Efficiency

Efficiency is defined as 'a set of attributes that bear on the relationship between the

level of performance of the software and the amount of resources used, under stated conditions'[Bach94]. Efficiency can be assessed by comparing the response time against performance requirements.

1.1.2.6 Maintainability

Maintainability is defined as 'a set of attributes that bear on the effort needed to make specific modifications'[Bach93]. This characteristic consists of many different attributes.

Some of the attributes can be measured statically such as modularity, structured-ness, static coupling, cohesion of modules and documentation. Where as the following attributes can only be measured dynamically through experimental and field-testing:

- Effort needed to modify a system due to changes of customer requirements.
- Testability to validate a modified software.

There are some attributes of maintainability that fall in different category of software metrics such as complexity, which is mentioned in the next section. These attributes can not be classified under a single metric type because they are common attributes to many other metric types, for example, modularity; a modular system is more maintainable and less complex than a monolithic system.

1.1.3 Complexity Metrics

Complexity metrics describe the relationships and the compositions of software

components. Complexity is not equivalent to size as may be claimed. It has to do with the number of relationships between elements of a set and the number of relationships between the sets of components [Sned95]. Complexity consists of attributes that can be considered to be maintainability attributes. Such as coupling and cohesion which are explained in next section.

1.1.4 Coupling

Coupling is defined as a measure of the degree of interdependence between modules [Fent91]. It is clear from the definition that coupling is a common attribute of complexity and maintainability. A highly coupled module indicates more complexity and is difficult to maintain. Coupling mainly is an attribute of a pair of modules, however global coupling can be derived from such attribute. Although coupling is defined as a measure, there are no standard numerical characterizations of this attribute [Fent91].

There are six main types of coupling defined by Myer reviewed in the literature ordered from worst to best coupling [Fent91].

If coupling is defined as a binary relations on pair of modules x and y, then:

Content coupling relation R_5: (x, y) $\in R_5$ if x refers to the inside of y, i.e. it branches into, changes data, or alters a statement in y.

Common coupling relation R_4: (x, y) $\in R_4$ if x and y refer to the same global data.

Control coupling relation R_3: (x, y) $\in R_3$ if x passes a parameter to y with the intention of controlling its behavior.

Stamp coupling relation R_2: (x, y) $\in R_2$ if x and y accept the same record type as a parameter.

Data coupling relation R_1: (x, y) $\in R_1$ if x and y communicate by parameters, each one being either a single data element or a homogeneous set of data items which do not incorporate any control element.

No coupling relation R_0: (x, y) $\in R_0$ if x and y have no communication, i.e. are totally independent.

1.2 Problem Definition

The purpose of this work is to:

- Provide a framework that acts as standard platform for automating the calculation of coupling metrics measurements.
- Provide a framework that acts as standard platform to facilitate the analysis of software system characteristics.
- The framework should make evaluations of different coupling metrics fair and unbiased toward any specific language or operating system.
- The framework should be language and operating system independent.
- The framework should satisfy software metrics assessments by automatically collecting, storing and analyzing software.

1.3 Thesis Organization

Chapter 2 presents a sample of coupling metrics proposed in the literature. The input parameter and the calculation methods of such metrics are also presented.

Chapter 3 presents the design of the system and how attributes and factors are considered while deriving the system design. Chapter 4 explains the implementation part of the work. Chapter 5 presents some study cases of C programs. Conclusion and future work are discussed in chapter 6.

Chapter 2

Coupling Metrics - Background

The definition of coupling mentioned in the previous chapter does not specify exactly the attributes and the method needed to calculate the coupling of a system. Therefore, there are many coupling metrics available in the literature that claim to measure total coupling of a system. In this chapter four metric methods will be presented to illustrate how each metric method calculates the system coupling, and what are the attributes that each method considers for deriving the global system coupling.

The following methods will be presenesented in this chaper:

Fenton and Melton, Offut, Harrold and Kotle, Dhama's and Cover Coefficient methods.

The last two methods, Damma's and Cover Coefficient are believed to have considered most system attributes that affect the system coupling. Such attributes are believed to play a major part in deriving system coupling.

2.1 Fenton and Melton Method

Fenton and Melton [Fent90], refined Myers [Fent91] coupling classification and assigned numbers in order to identify them. They proposed a method to measure the system coupling.

They assigned numbers are shown in Table 1 .

Coupling Type	Coupling Level	Modified Definition between Modules x and y
Content	5	Module x refers to the inside of module y i.e., it changes data, or alters a statement in y.
Common	4	Modules x and y refer to same global data.
Control	3	Module x passes a controlling parameter to y.
Stamp	2	Modules x and y accept the record type as parameter.
Data	1	Modules x and y communicate by parameters, each of which is either a single data or homogenous structure that does not incorporate control element.
No Coupling	0	Modules x and y have no communication, i.e., are totally independent.

Table 1. Fenton and Melton Modified Definition for Myers Coupling Levels.

The coupling measure they proposed depends on the following:

- The highest coupling level *i* exists between any two modules.
- The number of interconnections *n* between the modules i.e. number of parameters passed and/or global variables used by the two modules, etc.

The coupling of any pair of modules x and y C(x,y) is defined as the sum of the highest coupling level and the ratio of the number of their interconnections to the number of their interconnections plus 1. Mathematically,

$C(x,y) = i + n / (n + 1)$, where:

i = highest coupling level (worst coupling found in x, y).

n = number of interconnections between x and y.

Example 2.1

var flag : boolean;

```
Procedure x;

var local_int : integer;

var local_bool: boolean;

flag = false;

y( local_int, local_bool);

End.


Procedure y( p_1 : integer , p_2 :boolean);

.

flag = true;

End.
```

Figure 1. Example of Fenton and Melton method.

In example 2.1 in figure 1 procedures x and y uses same global variable, procedure x passes one int and one boolean arguments to procedure y.

Hence, the coupling level numbers used between procedure x and procedure y:

Flag=4;	coupling type = common.
Local_boolean=3;	coupling type= control .
Local_int = 2;	coupling type= data.

The maximum coupling level number is 4.

The interconnections between the modules are:

1. Global variable flag.

2. Parameter p_1.

3. Parameter p_2.

Total number of interconnections = 3;

The coupling of modules x and y :

$C(y,x) = i + (n / (n+1)) = 4 + (3 / (3+1)) = 4.750.$

Fenton and Melton proposed the *global coupling* to be the median of all pairwise module couplings.

2.1.1 Coupling Attributes

The following summarizes the attributes and factors that Fenton and Melton method considers:

- Global variables and their types.
- Local variables.
- Parameters.
- Function/Procedure calls.
- Number of read, write and control operation of each local, global and parameter made by each function.

2.2 Offut, Harrold and Kotle Method

Offut, Harrold and Kotle proposed a general coupling method based on data binding,

data slicing and coupling levels. They redefined Myers coupling classifications and added six coupling categories.

The redefined classes with numbers asigned to them are shown in Table 2.

Coupling Type	Coupling Level	Modified Definition between Modules x and y
Content	6	Module x refers to the inside of module y i.e., it changes data, or alters a statement in y or vice versa.
Common	5	Modules x and y refer to same global data.
External	4	Modules x and y communicate through an external medium such as a file.
Control	3	Module x passes a controlling parameter to y that is used to control the internal logic of y.
Stamp	2	Modules x and y accept the record type as parameter.
Data	1	Modules x and y pass data through scalar or array parameters.
No Coupling	0	Modules x and y have no communication, i.e., are totally independent.

Table 2. Offut, Harrold and Kotle Modified Definition of Myers Coupling Level.

The Extra six categories of coupling are:

- Call coupling: Occurs when module x calls module y without passing any arguments or share any data.
- Stamp coupling: Occurs when data flow from module x that calls a chain of intermediate modules in which data is passed to the last module y. This coupling is defined on multiple modules.
- Stamp/control coupling: Occurs when module x calls module y passing a data structure used as control.
- Stamp/data control coupling: Occurs when module x calls module y passing a data structure containing some data items used as controls.

- Non-Local coupling: A common coupling in which a variable is used by a number of modules.
- Global coupling: A common coupling in which a variable is used by all modules in the system.

The coupling measure depends on the maximum coupling level existing between any two modules and the number of interconnections of coupling between the pair of modules minus a constant.

It is similar to Fenton and Melton measure except that it substracts ½ . This is to emphasize that the effect of the number of interaction by intuition should be closer to level i more than to level i +1.

Mathematically $C(x,y)= i + (n -1) / (2* (n + 1))$.

2.2.1 Coupling Attributes

The following summarizes the attributes and factors that Offut, Harrold and Kotle method considers:

- Global variables and their types.
- Local variables.
- Parameters.
- Function/Procedure calls.
- Number of read, write and control operation of each local, global and parameter made by each function.

2.3 Dhama's method

Dhama proposed coupling methods that measure coupling of modules [Dham95]. A module defined as a compilation unit of code, which has function(s), procedure(s) and global variables.

He has categorized coupling into four categories:

- Data flow coupling caused by the parameters at the interface.
- Control flow coupling caused by the parameters at the interface.
- Global coupling caused by global variables.
- Environmental coupling caused by calling and being called by other modules.

He has assumed the following assumptions:

- Parameters or shared variables can be used as data or control variables.
- Control variables are assumed to have double coupling strength than data variables.
- Parameters and global variables have same coupling strength.

Variables used for diagnostic or error recovery variables are assumed as control variables.

Dhama proposed the following quantitative model for the four types of coupling:

Let:

C= total module coupling.

For data and control flow coupling:

i_1 = in data parameters.

i_2 = in control parameters.

u_1 = out data parameters.

u_2 = out control parameters.

For global coupling:

g_1 = number of global variables used as data.

g_2 = number of global variables used as control.

For environment coupling:

w= number of modules called.

r= number of modules calling the module under consideration.

Dhama summed up all above attributes into one, called m, which is defined as:

$$m = i_1 + q_6 i_2 + u_1 + q_7 u_2 + g_1 + q_8 g_2 + w + r .$$

where q_6, q_7 and q_8 are constants and assumed to be 2 as first heuristic estimate.

The minimum value of r is 1.

The coupling of the system C is inversely proportional to m to point out that low coupling will be indicated by high numbers, because increasing the numbers are generally associated with increasing goodness.

$C \propto \frac{K}{m}$ where k is a constant, assuming k =1, then

$C \propto \frac{1}{m}$.

Dhama used the above method to calculate manually the coupling of 2 modules achieving the same functionality, which is sorting an array. The first module

implemented as single procedure while the second as four procedures. The second module has shown better results than module one. This result agrees with intuition that four smaller procedures are easier to understand and have less coupling than single monolithic procedure implementation.

2.3.1 Coupling Attributes

-The following summarizes the attributes and factors that Dhama's method considers:

- Number of input data parameters
- Number of output data parameters.
- Number of input control parameters.
- Number of output control parameters.
- Number of global variables used as data.
- Number of global variables used as control.
- Number of modules called by the module under consideration.
- Number of modules calling the module under consideration.

2.4 Cover Coefficient Concept

Based on hypothesis known as "clustering hypothesis" which states that "closely associated documents tend to be relevant to the same request", Can and Ozkarahan

[Can90] have developed a cover coefficient matrix that is used to cluster documents in information retrieval system. They developed a seed based algorithm, which mainly depends on the idea that the seed must attract relevant documents onto itself.

Using the cover coefficient concept, documents relationship of coverage and similarities are reflected in so called cover coefficient coupling matrix (C), whose elements convey documents/term coupling of a symmetrical as well as an asymmetrical nature.

They defined a matrix called D-Matrix or definition matrix which basically represents the document database $d=\{d_1, d_2, \ldots, d_m\}$ described by the index terms $t=\{t_1, t_2, \ldots t_n\}$.

The cover coefficient coupling matrix or C matrix, is a document-by-document matrix whose entries $c_{ij}\ (1 \le i, j \le m)$ indicate the probability of selecting any term of d_i from d_j.

The C matrix indicates the relationship between documents based on a two-stage probability experiment. The experiment randomly selects terms from documents in two stages. The first stage randomly chooses a term t_k of document d_i; then the second stage chooses the selected term t_k from document d_j.

The entries of definition matrix D, d_{ij} $(1 \le i \le m,\ 1 \le j \le n)$ must satisfy the following conditions.

- Each document must have at least one term.
- Each term must appear at least in one document.

From the definition of cover coefficient, the entries of the C Matrix c_{ij} can be calculated by first selecting arbitrary term of d_i, say t_k, and use this term to try to

select document d_j, in other words that is to check if d_j contains t_k , i.e. two stage method.

Mathematically:

Let $S_{ik} = d_{ik} \times \left[\sum_{h=1}^{n} d_{ih}\right]^{-1}$,

$S'_{jk} = d_{jk} \times \left[\sum_{h=1}^{m} d_{hk}\right]^{-1}$,

for $1 \le i,\ j \le m,\ 1 \le k \le n$.

The C matrix then can be formed from the metrics named S and S', that is

C = $S \times S'^{T}$ (S'^{T} indicates the transpose of matrix S') , and the matrix elements are S_{ik} and S'_{jk} defined above.

S_{ik} and S'_{jk} indicate the probability of selecting t_k from d_i and the probability of selecting d_j from t_k respectively. Hence using S and S', the entries of C matrix can be defined as follows:

$$c_{ij} = \sum_{k=1}^{n} S_{ik} \cdot S'^{T}_{kj}$$

$= \sum_{k=1}^{n}$ (Probability of selecting t_k from d_i) **X** (Probability of selecting d_i from t_k).

Where $S'^{T}_{kj} = S'_{jk}$, This can be rewritten:

$$c_{ij} = \alpha_i \sum_{k=1}^{n} d_{ik} \beta_k d_{jk}\ ,\ \text{for}\ \ 1 \le i,\ j \le m,.$$

Where $\alpha_i and \beta_k$ are the reciprocals of the ith row and the kth column sum, as shown below:

$$\alpha_i = \left[\sum_{j=1}^{n} d_{ij} \right]^{-1} , \; 1 \le i \le m .$$

$$\beta_i = \left[\sum_{j=1}^{m} d_{jk} \right]^{-1} , 1 \le k \le n .$$

The cover coefficient matrix has the following properties:

(1) For $i \neq j, 0 \le c_{ij} \le c_{ii}$ and $c_{ii} > 0$.

(2) $c_{i1} + c_{i2} + \ldots + c_{im} = 1$ (i.e., sum of row i is equal to 1 for $1 \le i \le m$,).

(3) If none of the terms of d_i is used by other documents, then $c_{ii} = 1$;otherwise, $c_{ii} < 1$.

(4) If $c_{ij} = 0$, then $c_{ji} = 0$, and similarly, if $c_{ij} > 0$, then $c_{ji} > 0$, but in general, $c_{ij} \neq c_{ji}$.

(5) $c_{ii} = c_{jj} = c_{ij} = c_{ji}$ iff d_i and d_j are identical.

Example 2.2 :

Consider 3 documents containing 5 terms, the definition matrix or D matrix is represented as follows:

$$D = \begin{array}{c} \\ d1 \\ d2 \\ d3 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

Table 3. Definition matrix of the example 2.2.

C=	d_1	d_2	d_3
d_1	0.3	0.5	0.2
d_2	0.1	0.3	0.6
d_3	0.2	0.4	0.4

Table 4. Coupling matrix of the example 2.2.

The weights of cover coefficient matrix, as shown in example 2.2, use a binary representation for present/absent of terms in documents database. This is sufficient for document clustering in information retrieval, since term's relationships are flat rather than hierarchical[Algh94]. However, the method is extended to have weights for the elements participating in the definition matrix, as explained in the next section.

2.4.1 Al-Nasser Weighing Scheme

Based on cover-coefficient concept of measuring couplings among modules, Al-Nasser has proposed a new weighing scheme that can be used in conjunction with cover-coefficient concept to measure the coupling of modules of a C program [Alna97].

Based on the fact that a system consists of objects that grouped together to form working system. These objects are connected together through variable bond strengths.

They are four types of bonds identified [Algh95]:

Shared External Resources: These resources are part of the environment the system is using.

Interface Resources: These resources that facilitate the communication and the movement of information between different components of the system.

Common Elements: These are the elements that are part of two or more objects.

Inclusion: These are objects that are totally included in another object.

A system can be mapped into the definition matrix through a repetitive decomposition of system objects and their consisting elements until the system elements are simple and can not be further decomposed.

The definition matrix rows will represent then objects, where as the definition matrix columns will represent basic elements or properties of objects. The entry d_{ij} represents the weight of the jth element in the ith object.

Clearly having the definition matrix above, a coupling matrix can be easily calculated. Then entry c_{ij} of coupling matrix measures the coverage extent of the ith object in the jth object as well as the separation extent of the ith object from other objects. In order to obtain weight for elements participating in the definition matrix, Al-Nasser has considered three main interface items that cause coupling between modules. The third one is excluded because C language does not permit nested procedures or functions.

These interface items are:

1. Parameters;
2. Global data;

Inclusion.

The weighing scheme Al-Nasser proposed was based on Myers classification of coupling mentioned in section 1.3.1.

Excluding content coupling,

- Common coupling is caused by global data.
- Control coupling is caused by the use of the interface item to control the logic of one or two pairs of modules.
- Stamp coupling is caused by non-homogenous interface data items, i.e., structure and union.
- Data coupling is caused by using homogeneous or simple interface element. Also data coupling is caused by not using any of global, control or stamp interface items. The data coupling is the lowest coupling between any pair of modules.

Using above categorization as dimension for each interface item. Al-Nasser has classified these dimensions as flags to indicate whether the interface item is global, used for control/data or is a structure/scalar .

Table 5. shows the proposed weight for each possible combination:

Parameter(0)/ Global(1)	Data(0)/ Control(1)	Non-Structure(0)/ Structure(1)	Proposed Weight
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

Table 5 Al-Nasser proposed weight of coupling.

From table 5 the weights of interface items include the three factors that Myers proposed. The factors are:

- *Element access scope*: The global access scope weight is 4 while the parameter access scope weight is 0.
- *Element access purpose*: The control access purpose weight is 2 while the data access purpose weight is 0.
- *Element type*: The structure type weight is 1 while the non-structure type weight is 0.

Based on the weight obtained, the type of coupling is identified.

For example:

- When weight is 1 then data coupling is identified.
- When weight is Even then stamp coupling is identified.
- When weight is 3,4,7 or 8 then control coupling is identified.
- When weight is Greater than 4 then global coupling is identified.

The calculation of weight of the interface elements in the definition matrix is performed as follows:

Let there be n interface elements, (i.e., parameter, global, or return items from a function), in a system of m modules, where

M_i: the ith module in the system of m modules,

G_j: a flag stating whether the jth interface is global(1) or parameter(0).

C_j: a flag stating whether the jth interface is control(1) or data(0).

S_j: a flag stating whether the jth interface is Structure(1) or Non(0).

Then

$d_{ij} = d_{ij} + \sum (4 * G_j + 2 * C_j + S_j + 1)$ when M_i is the called module and

$d_{ij} = d_{ij} + \sum (4 * G_j + 2 * C_j + S_j + 1)$ when M_i is the calling module, where

d_{ij} is the coupling weight of the jth interface element in the ith module or the definition matrix entry of the jth interface element in the ith module. The initial value of d_{ij} is zero.

Using the above entries of the definition matrix Al-Nasser was able to manually calculate the coupling matrix of a sample set of C programs.

2.4.2 Coupling Attributes

The following summarizes the attributes and factors that Cover-Coefficient method considers:

- Global variables and their types (structures or scalar).
- Local variables and their types (structures or scalar).
- Formal parameters and their types (structures or scalar) and their positions in the parameter list.
- Function/Procedure calls and their sequence (caller and called).
- Arguments passed and their positions in the call.
- Control variables (for global and local).

- Control Parameters.
- Function names and their return types.
- Data types (structured or scalar).
- Number of write, read, control and passed-parameter operations, of each global, local and parameter made by each function.

2.5 Motivation of the work

Coupling metrics presented in the literature are not general, and they have many drawbacks:

- They are designed to suit specific environment.
- They do not measure all the system attributes.

Due to the absence of standards the following points are the main motivation of this thesis:

- There is a need to provide a standard and general platform to examine different coupling metrics at once and compare them on equivalent basis. The platform should be language and environment independent.
- There is a need to provide a framework to automate the calculation of different coupling metrics.
- Any new proposed coupling metrics calculation should be easily implemented and validated.

- Some coupling metrics perform well on certain set of programs, and very poorly on the other set of programs suited for other metrics. We need to identify weak and strong points of each method.
- Many coupling metrics proposed in the literature are not subjected to large production program. An automatic way to provide large set of data to these metrics will validate their applicability.
- Refinement of algorithms to calculate coupling metrics should be made easy and fast to develop.

Chapter 3

The Framework of Couplimg Measurement System Design

Coupling metrics mentioned in previous chapter showed how each coupling method is calculated. Each coupling method tackles the system coupling from different prospective and considers different attributes. Hence, the resultant coupling of each method is different.

The calculation procedures for coupling metrics are done either manually or through automatic tools that are made to measure the coupling metrics for a specific language under specific operating system. Therefore, it is very difficult and unfair to compare different coupling metrics. The comparisons and the evaluations will be subjective issues.

In this thesis a new framework is presented to provide a standard platform for coupling metric measurement. The framework leverages the system constructs into a more abstract level. It captures the necessary attributes and factors while hiding the details of the language or the operating system. The attributes captured should be stored in a system repository. The coupling methods will retrieve their input attributes from the system repository in order to carry out the measurement.

Therefore, any evaluation or comparison of different metrics is not done in isolation, and a fair comparison may be achieved.

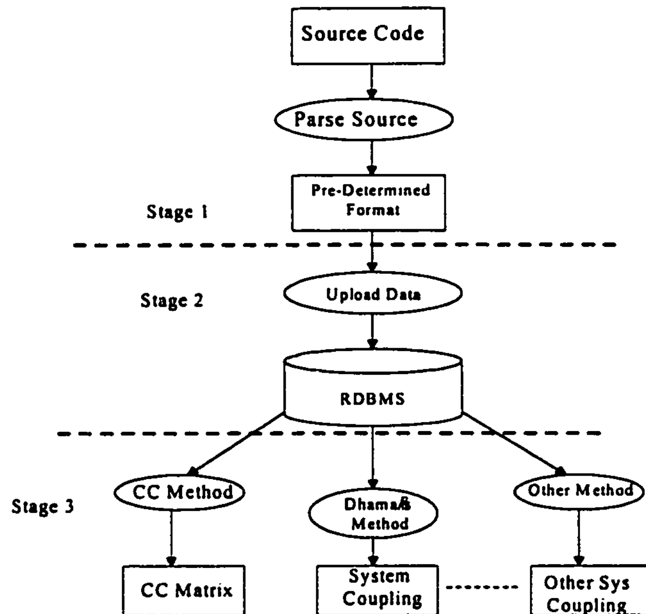


Figure 2. The three system stages

3.1 System Stages

The framework consists of three main stages. Each stage can be developed and implemented independently as shown in Figure 2.

The following are brief descriptions of the roles and the functionality of each stage. Note that function and procedure are referred interchangeably.

3.1.1 Stage One

This stage involves parsing the system to be analyzed to extract its factors and attributes. These attributes are in turn fed into second stage.

The parsing tools therefore, are language and operating system dependent. But the output of these tools for all imperative languages must have the same format and data

structures to be understood by stage two. The format and the data structures expected by second stage are discussed in details in chapter 4.

3.1.2 Stage Two

This stage involves populating the system attributes provided by previous stage into relational database. The populating tools are provided by the database system.

Special tools are developed to identify system calls and create system functions with proper number of parameters. The tools are developed in Progress 4GL and access the database to create new records that represent system functions.

The created system functions records in the database are provided to help coupling methods to measure the effect of system calls on the program under-consideration.

The schema of the database of stage two represents the abstract format of the system constructs of the imperative languages, and database itself is the system repository where all coupling attributes of the systems under consideration are saved.

3.1.3 Stage Three

This stage involves the development of modules to calculate desired coupling metrics. The modules should access the databases to retrieve the required system attributes.

It is necessary to implement each coupling metric by developing a module that accesses the database. Hence, every develeoped module, that represents a metric method, will have the same set of data and environment. Therefore, comparison of

different coupling metrics is made fair and easy.

A complete system based on the above has been implemented to illustrate the applicability of the new framework and to obtain empircal results of production programs. The system consists of special C-language parser, commercial Relational Database Management System (Progress RDBMS) and modules to calculate few coupling metrics developed in Progress 4GL.

3.2 Development Phases of the Abstract Format

There were four phases carried out to produce the design of the abstract format provided in stage two.

Phase 1:

Identify the attributes and the factors considered by each coupling method. The followings are attributes that are found in all coupling methods:

- Data types.
- User defined types for structured data types.
- Global variables and their types.
- Local variables and their types.
- Formal parameters and their types.
- Function/Procedure names.
- Function's return type.

- Global variables used inside function/procedure body.
- Number of write operations in the function body for each parameter, global or local variable (i.e., assignment-to var is at LHS).
- Number of read operations in the function body for each parameter, global or local variable (i.e., var is at RHS).
- Number of times each parameter, global or local variable used as control variable in the function body.
- Number of times each parameter, global or local variable is passed as an argument in function calls.
- Function/Procedure invocations and the sequence of calls.
- Name of passed arguments in each function call done by each function.

Phase 2:

Identify the attributes that are derivable from other attributes.

The following attributes can be derived from other attributes in the system:

- Parameter aliases of global-to-formal type, i.e., swap(x, y) where x or y are global variables.
- Parameter aliases of formal-to-formal type, i.e.,

```
void print_max( int n){
max(n);
...
}
```

n is a formal parameter.

- Parameter aliases for local-to-formal types, i.e. max(x,y) where x or y are local variables.
- Chain of function calls i.e., to find a call path from function 1 to function n.

Phase 3:

Identifying the relations between attributes found in phase one and phase two in order to define storage model. The relations found for imperative languages constructs are the following:

- A system *has* **a** number of modules.
- A module *has* **b** number of global variables.
- A module *has* **m** number of functions.
- A function *has* **p** number of formal parameters.
- A function *has* **l** number of local variables.
- A function *uses* **g** number of global variables.
- A function *calls* **x** number of functions.
- A function *passes* **y** number of arguments in each call.
- Each argument passed *is* a variable, parameter, expression or a function call.
- A function *writes to* (modifies) each used global, local or parameter, **w** number of times.
- A function *reads from* each used global, local or parameter, **r** number of times.

- A function *uses as control* each used global, local or parameter, **c** number of times.
- A function *passes* as argument to other functions, each used global, local or parameter, **s** number of times.
- A function *has* **u** number of statements.
- A global, local, parameter or a function *has/return* data type.
- A data type *is* either simple or complex data type
- Complex data *has* fields.
- A field *is* either simple or complex data type.
-

Phase 4:

Map the relationships of attributes found in phase 3 into standard storage model. The verbs in Italics described in phase 3, are the relations between tables (entities). The operations made by functions such as writing, reading, controlling or passing arguments are attributes (in terms of relation model) of each global, local or parameter entity. The relationships of attributes found map easily into Entity-Relationship diagram (see Figure 3. E-R Diagram of the system). The E-R diagram in turn, is converted into relational database schema. Therefore, any system under-consideration can be saved and retrieved in a relational database management system (RDBMS).

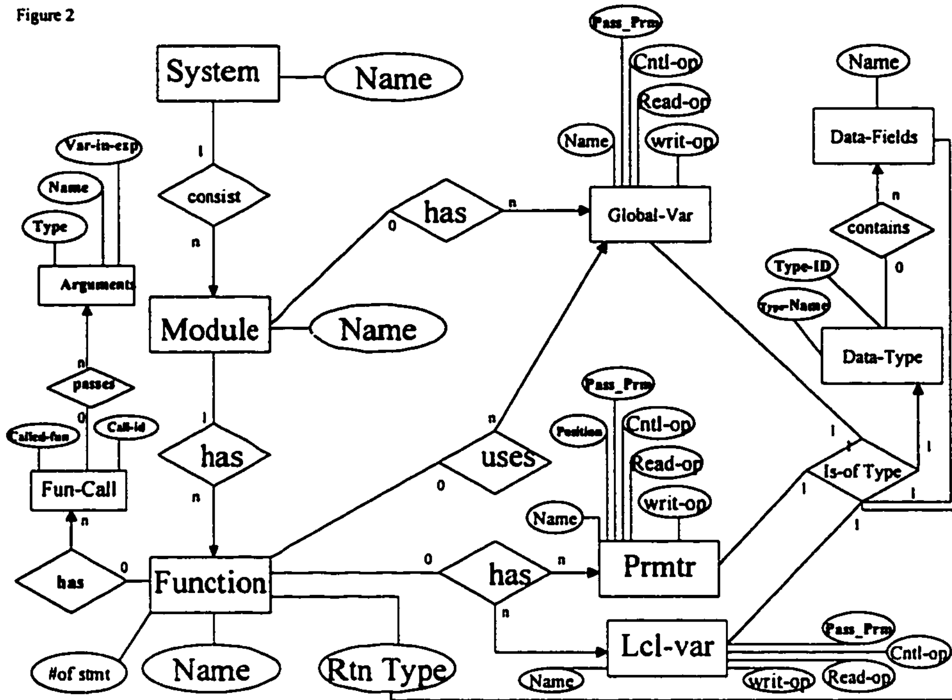


Figure 3. E-R Diagram of the system

The E-R diagram of the proposed model is converted into relational schema. The final normalized schema is described in the rest of this subsection.

Note that:

- x + y means x is concatenated to y to form xy, and
- attributes mentioned below refer to the fields of the table.

Table : **system**

Primary key: s_name. (unique)

Attributes	Type	Description
S_name	Character	System name

Table 6. Schema definition for table system.

Table : **module**

Primary key: s_name + m_name. (unique)

Attributes	Type	Description
M_name	Character	Module name
S_name	Character	System name

Table 7. Schema definition for table module.

Table : **function**

Primary key: fun_id. (unique)

Attributes	Type	Description
Return type	Integer	The type that the function returns
Fun_id	Character	Introduced for efficiency. It is just S_name + m_name + f_name.
Number_of_statement	Int	Num of statement in the function body excluding local declaration.
F_name	Character	Function name
M_name	Character	Module name containing this function
S_name	Character	System name containing this function

Table 8. Schema definition for table function.

Table : parameter

Primary key: fun_id + position . (unique)

Attributes	Type	Description
Parameter_ type	Integer	The type id of the parameter
Fun_id	Character	The fun_id of the parent function.
Position	Int	The position of the parameter in the parameter list of the parent function.
P_name	Character	Parameter name
Read-op	Int	Number of read operation made on this parameter inside the function body.
Write-op	Int	Number of write operation made of this parameter

		inside the function body.
Control-op	Int	Number of times the parameter used in control logic inside the function body.
Pass_parm	Int	Number of times the parameter is passed as argument to another function inside the function body.

Table 9. Schema definition for table parameter.

Table : global_var_used

Primary key: fun_id + g_name . (unique)

Attributes	Type	Description
Var_ type	Integer	The type id of the global variable
Fun_id	Character	The fun_id where the global variable is used.
g_name	Character	Global variable name
Read-op	Int	Number of read operation made on this variable inside the function body.
Write-op	Int	Number of write operation made on this variable inside the function body.
Control-op	Int	Number of times the variable used in control logic inside the function body.
Pass_parm	Int	Number of time the variable is passed as argument to another function inside the function body.

Table 10. Schema definition for table global_var.

Table : local_var

Primary key: fun_id + l_name . (unique)

Attributes	Type	Description
Var_ type	Integer	The type id of the global variable
Fun_id	Character	The fun_id where the local variable is defined.
L_name	Character	Local variable name
Read-op	Int	Number of read operation made on this variable inside the function body.

Write-op	Int	Number of write operation made on this variable inside the function body.
Control-op	Int	Number of times the variable used in control logic inside the function body.
Pass_parm	Int	Number of time the variable is passed as argument to another function inside the function body.

Table 11. Schema definition for table local_var.

Table : data-types

Primary key: s_name + m_name + f_name + type_id . (unique)

Attributes	Type	Description
Type_id	Integer	The type id number
Type_name	Character	A string describing the type name
F_name	Character	Function name where the type is declared. Empty if type is declared in global scope.
M_name	Character	Name of the module where the is declared.
S_name	Character	Name of the system containing the module where the type is declared.

Table 12. Schema definition for table data-type.

Table : type-field

Primary key: s_name + m_name + f_name + parent_type_id + field_name . (unique)

Attributes	Type	Description
Type_id	Integer	The type id of the field
Field_name	Character	The name of the field.
Parent_type_id	Int	The type id of structure containing this field.
F_name	Character	Function name where the parent structure is declared.
M_name	Character	Name of the module where the parent structure is declared.
S_name	Character	Name of the system containing the module where the type is declared.

Table 13. Schema definition for table type-field.

Note:

The attributes s_name, m_name, f_name, parent_type_id are used as a key to identify the structure containing this field.

Table : **fun-call**

Primary key: call_id . (unique)

Attributes	Type	Description
Fun_id	Character	The function id of the caller.
Called	Character	Name of called function.
Seq	Int	Call sequence
Call_id	Int	Each call has unique call id in the system. Call id is introduced for efficiency and simplicity. It can be replaced by (fun_id + called + seq).

Table 14. Schema definition for fun-call.

Table : passed-argument

Primary key: call_id + position . (unique)

Attributes	Type	Description
Argument_ type	Character	The value of this attribute should describe the argument type i.e., *VARIABLE, EXPRESSION, PARAMETER, FIELD, FUNCTON_CALL and CONSTANT* etc.
call_id	Int	This field is meant to identify what call sequence this argument is passed for.

Position	Int	Position of the argument in the function call.
Argument_name	Character	If argument_type is a parameter, a variable or a function, this attribute contains the name, otherwise is empty.
Var_in_exp	Char	If argument type is an expression, this attribute should have all variables, parameters participated in this expressions. They are separated by comma.

Table 15. Schema definition for passed-argument.

The following tables are the system functions and their parameters created after populating the user functions in stage two.

Table : sys-fun

Primary key: s_name + f_name. (unique)

Attributes	Type	Description
Return type	Integer	The type that the function returns Value is set to 1.
Fun_id	Character	This is s_name + f_name .
F_name	Character	System function name
S_name	Character	System name containing this function

Table 16. Schema definition for table sys-fun.

Table : sys-parameter

Primary key: fun_id + position . (unique)

Attributes	Type	Description
Parameter_ type	Integer	The type id of the parameter . Value is set to 1.
Fun_id	Character	The fun_id of the parent sys-function.
Position	Int	The position of the parameter in the parameter list of the parent function.

P_name	Character	Parameter name, usually Pi, Where i= 1..n, n is last position number of this sys-function.
Read-op	Int	Number of read operation made on this parameter inside the function body. Value is set to 0.
Write-op	Int	Number of write operation made of this parameter inside the function body. Value is set to 0.
Control-op	Int	Number of times the parameter used in control logic inside the function body. Value is set to 0.
Pass_parm	Int	Number of time the parameter is passed as argument to another function inside the function body. Value is set to 0.

Table 17. Schema definition for table sys-parameter.

The above tables showed how programming construct of imperative language are mapped into relational model without sacrificing the required details to measure coupling metrics.

3.3 Benefits of the FCMS

- The system satisfies metric assessment requirement (collecting, storing and analyzing) [Bach94].
- It provides a standard platform for evaluating and comparing coupling methods.
- It hides the details of the languages and operating systems from stage two onwards.
- Modules developed for one coupling metric, i.e. Dhama's method, to analyze a system written in C-Language, need not to be modified if other system written in different language, i.e. Pascal, is populated into the database.

- If a new coupling metric method is proposed, only its calculating module need to be implemented, and same set of data might be used.
- Since modules are usually developed in 4GL, further refinements of algorithms surveyed in the literature are made easy.
- Due to the fact that software systems are represented by a relational model and saved as data entities in the database, the properties and characteristics of such systems are easier to obtain and to analyze.

Chapter 4

System Implementation

The coupling measurement system developed in this thesis has three stages. Implementations of the first and the last stages are explained in this chapter. The difficulties and design decision are mentioned. Stage two consists of two main parts, first part consists of system utilities provided by the RDBMS vendor to populate data into the database, which could differ from one database vendor to another. The second part of stage two consists of utility program developed for this thesis to identify system functions. This utility program is explained in section 4.2.2. The source code listing of the utility program is included in appendix F.

4.1 Stage one implementation

The first part of the general coupling measurement system developed in this thesis involves parsing a source code to extract the system attributes. Upon the completion of the parsing phase, the parsing tools write these attributes in pre-determined file format to be populated into the database in the second stage.

C-language is a very popular language and good candidate for coupling measurement. There are a lot of production programs written in C that are available and need to be

analyzed and studied. Therefore, it was decided that the special purpose parser should parse C-language source code.

The second decision was either to develop new parser from scratch or modify production compiler to produce the desired output. The latter choice was chosen. The reasons for such a choice are:

Production compiler is more reliable and guaranteed to work than developing new parser from scratch.

No need to proof that the parser conforms to syntax and semantics of C-Language.

Parse trees produced for system under consideration will be the same in actual compiling phase and in the phase of extracting attributes. Hence, expression evaluations and function calls are identical.

- Unusual constructs of C-language will be handled the same way as the compiler does i.e., +x= which is equivalent to x=x + 1.
- Error checking and type conversions are done automatically.
- Production compilers can compile old style source code and ANSI standard source code.

Clearly, the advantages of using production compiler out weigh the disadvantages. Even though the compiler undergoes many phases not needed for the model such as optimization, expression reduction, translation into assembly etc, the implementation of the points mentioned above will be a major work, susceptible to error and unnecessary re-work of system that is available.

4.1.1 System Attributes Extraction

In order for the parser to satisfy stage two requirements, the compiler was extended to carry out some actions at certain points during parsing phase.

The purpose of these actions is to identify language constructs and attributes, process them, then save the result in the memory for final output.

The following points describe where the extended actions are inserted and how they function.

Note that "system" refers to FCMS, "compiler" refers to production compiler.

4.1.1.1 Type declaration

Each type has a type name and unique type-id. The compiler creates the type-id that is unique for each module. The type name is string describing the type, which is a requirement of stage two.

The system has a list containing all type names defined so far, along with their unique type-id.

On parsing type declaration, the system checks if unique type-id is available in the list, if not, it translates the type into string and then adds it to the list with type-id. This way no duplicate type-ids are inserted into the list.

For user-defined types; the type name, its unique type-id, function name where the definition occurred and the file containing the definition are recorded.

The function name is recorded to distinguish different types of the same name defined in different places in the system, as shown in Figure 4.

```
typedef  char my_type;
    my_type x;
    ....
    int max( char name){
    typedef  long double my_type;
my_type y;
    ....
    }
```

Figure 4. Type name aliases.

Clearly variables y and x have different types.

If the recorded function name is blank then user-defined type has a global scope.

If user defined type is struct or union type, the name of the struct/union is assumed to be the type name. The fields of the struct/union are recorded separately in field list. The field list contains field name, field type-id, and type-id of the container struct/union. The field that itself is struct/union is handled the same as scalar one. Because the inner most struct/union would have already been parsed and added to the type list as in Figure 5.

```
Struct employee {
    Int salary;
    Struct person {
            Int age;
            Char name[30];
```

```
    } figure ;
}
```

Figure 5. Nested complex data structures.

The parser would have added struct person to the type list with its type-id, before struct employee, therefore field figure will have its type-id already in the type list.

4.1.1.2 Functions declaration

Once the compiler has finished parsing the function header and before parsing the function body, the function name and the return type distinguished by unique type-id are recorded. A flag called start-of-function is set to 'yes' indicating start of function. On exit of function body the flag is set to 'no'.

This flag indicates the scope of parsing, so during the declaration of types or variables the extended action can determine the scope of variables for global and local scope.

4.1.1.3 Parameters declaration

Due to the fact that C language has two different ways of parameter declaration, old style and ANSI standard, each parameter declaration is treated differently.

Examples of old style and ANSI style parameter declaration are illustrated in figure 6 and figure 7 respectively.

```
char  swap ( src, dest, flag)
 float src;
float dest;
int flag;
{
/* function body */
}
```

Figure 6. Old style parameters declaration.

```
char  swap ( float src, float dest, float flag)
{
/* function body */
}
```

Figure 7. ANSI standard parameters declaration.

The compiler treats each style differently and produces different parse tree; special care was taken to handle each case separately. The parsing of parameters are done after finishing function declaration, hence, each parameter can be associated with its enclosed function. The parameter name, parameter unique type-id and the enclosed function name are recorded.

4.1.1.4 Global variables declaration

If start-of-function flag is set to 'no' the current variable declaration has a global scope otherwise it is local variable. The name of the variable, the unique type-id and

the file name containing the variable are recorded.

If the unique type-id is not found in the type list described above, then the system translates the type to string then adds it to the type list with type-id, empty string replaces the function name indicating type has been globally defined.

In case of user-defined type with no type tag as in figure 7, the variable name is considered to be the type name and added to type list. The process is the same as in section 4.2.1.

```
struct {
int age;
    char name[10];
} person;
```

Figure 8. User-defined data type with no type-tag

In figure 7, person is a variable of type name *person.*

Variables declared as extern are not added to global var list of the module, because they will be added to global var list of the module containing original declarations.

4.1.1.5 Local variables declaration

If start-of-function flag is set to 'yes' the current variable declaration has a local scope. The name of the variable, the unique type-id, the function name containing the variable and the file name are recorded.

If the unique type-id is not found in the type list described above, then the system translates the type to string then adds it to the type list with type-id ,the function name and file name.

4.1.1.6 Function Calls

C compiler treats function calls as expressions, hence returns the address of the returned values. The compiler calls the inner most function and passes the return address as an argument to the outer function one level above. These sequences appear as the main caller calls the inner most function, then the main caller calls one level above and so on.

Each function call the system assigns a unique call-id.

```
void main(int arg) {
int number;
number =10;

printf("hello number  %s \n", convert_to_string(multiply(number,arg)));
}
```

Figure 9. Functions calls sequence.

The compiler converts function calls in Figure 9. Functions calls sequence. into the following sequence:

- main calls multiply, passing variable number as first argument and parameter arg as second argument.
- main calls convert_to_string, passing return address of function multiply.
- main calls printf, passing constant string as first argument and the return address of convert_to_string as second argument.

This behavior is recommended because the called functions are not calling each other directly from their function body. Therefore any resultant coupling is done only through main function.

4.1.1.7 Passed Arguments

There are only five types of arguments passed in any function call.

- Variable argument type, scalar or struct.
- Parameter argument type, scalar or struct.
- Constant argument type.
- Function call i.e. address of return value.
- Expression.

Stage two of the FCMS requires the type of the argument and the name of any passed arguments.

For variables and parameters type:

The name and the type are recorded.

For constant type:

An empty string with constant type is recorded.

For function_call type:

The function name and function_call type are recorded .

For expression type:

An empty string with expression type is recorded, and all variables participating in the expression are recorded in an extra field. Variable names are separated by comma.

4.1.1.8 Variables that appear at LHS of assignment

If a global, local or a parameter appears in the LHS of an assignment then increment the write-op for that variable by one.

If a global, local or a parameter appears in post/pre expression then increment the write-op for that variable by one. Example, i++ or --i.

4.1.1.9 Variables that appear at RHS of assignment

If a global, local or a parameter appears in the RHS of an assignment then increment the read-op of that variable by one.

4.1.1.10 Variables that appear in logical expression

If a global, local or a parameter appears in any logical expression then increment the control-op of that variable by one.

Expressions appear in

if(expr) or switch (expr) or while(expr) or

do { } until (expr) or for (; expr;)

statements are assumed to be logical expression. All the variables participating in such expression have their control-op incremented by one.

4.1.1.11 Variables passed as arguments in function calls

If a global, a local or a parameter appears in a function call as an argument then increment pass_parm of that variable by one.

When a module is completely parsed, the system attributes mentioned above are written into nine flat format text files. Each file contains system attributes that represent one table in the database mentioned in chapter 3. The attributes of the database tables are comma separated in the text file and should be ordered in the same way as in the database.

For example, for table *function,* the representation in the text file is as follows:

1,"GCMSMOD1GET_MAX_NUM",14,"GET_MAX_NUM","MOD1","GCMS"

13, "GCMSMOD1GET_MIN_NUM", 11, "GET_ MIN_NUM","MOD1","GCMS"

.

.

.

5, “GCMSMOD1FORMA_NAME”, 14, “FORMAT_NAME”,”MOD1”,”GCMS”

the first number is return type-num the second string the function unique id (fun_id) then the number is number of statements followed by function name, module name and system respectively.

A complete listing of the FCMS functions is included in appendix B.

4.2 Stage two implementation

There are two main steps needed to be carried out in stage two:

4.2.1 Populating the system attributes into the database

The process of populating the system attributes into the database is achieved by using the utilities provided by the database management system. The utilities are fast and easy to use. However, it expects the format of data to be uploaded to be the same format as explained in stage one implementation.

4.2.2 Creating System calls and their formal parameters

In this process a utility program was developed to identify system calls from user function calls. Separate tables are used for system calls and their formal parameters.

The utility scans the call sequence of the system under consideration. If the called function is not in the function table then it is assumed to be a system call. Hence, it is added to the system function table if it does not already exist. On each reference to a system function, The number of its parameters is examined. If the number of parameters examined is greater than the recorded parameters of the system function, extra parameter records are created to match the maximum number of parameters of that function. This is done to handle system functions with variable number of parameters.

The source code listing of the utility program is included in appendix F.

4.3 Stage three implementation

This stage involves the development of a separate module for each required coupling method.

Dhama's method and Cover-Coefficient matrix were chosen for two reasons:

- They are not tested on large production programs.
- They consider the major factors in their calculation procedures i.e., element access scope, element access purpose.

4.3.1 Dhama's method

The original Dhama's method was designed to calculate the coupling of Ada programs, where the language syntax specifies the direction of the function

parameters i.e., input/output.

The direction of the parameters can be obtained by checking write-op, read-op or pass_parm of each parameter.

If write-op >0 then it is an output parameter.

If read-op > 0 then it is an input parameter.

If pass_parm-op > 0 then it is an input parameter.

Checking for control parameter is similar to direction check.

If control-op> 0 then it is a control parameter.

The algorithm used to calculate Dhama's method is shown in Figure 10.

```
/* define temporary table to hold the coupling attributes for each function **/

DEF TEMP-TABLE dhama-table
        field function-name             as character
        field input-data-parm           as int
        field input-control-parm        as int
        field output-data-parm          as int
        field output-control-parm       as int
        field num-of-global-data        as int
        field num-of-global-control     as int
        field num-of-called-mod         as int
        field num-of-calling-mod        as int
        field mod-coupling              as dec.

/* Algorithm is in pseudo language */

For each function of system-under-consideration:
    Create dhama-table.
    For each global variables used by this function:
        IF global-var.control-op > 0 THEN
            Increment dhama-table.num-of-global-control by one.
        ELSE
            Increment dhama-table.num-of-global-data by one.
    End.
```

```
For each fun-call where caller= current-function :
        Increment Dhama-table.num-of-called-mod by one.
End.
For each fun-call where called = current-function :
        Increment Dhama-table.num-of-calling-mod by one.
End.

For each parameter of this function:
        IF parameter.control-op >0  THEN DO:
                IF  parameter.read-op > 0 THEN
                        Increment dhama-table.input-control-parm by one.
                IF  parameter.write-op > 0 THEN
                        Increment dhama-table.output-control-parm by one.
        END.
        ELSE DO:
                IF parameter.read-op > 0 THEN
                        Increment dhama-table.input-data-parm by one.
                IF parameter.write-op > 0 THEN
                        Increment dhama-table.output-data-parm by one.
        END.

END.
Dhama-table.mod-coupling= 1 / ( dhama-table.input-data-parm + (2 * dhama-
table.input-control-parm) +   dhama-table.output-data-parm + ( 2 * dhama-
table.output-control-parm) + Dhama-table.num-of-global-data + (2 * dhama-
table.num-of-global-control) + Dhama-table.num-of-called-mod + dhama-
table.num-of-calling-mod.

End for outer most loop.

/** output the result **/

For each dhama-table:
        Disp mod-coupling (avg).
End.
```

Figure 10. Algorithm that calcultes Dhama's coupling metrics.

4.3.2 Cover-Coefficient method

Cover-Coefficient method is believed to have considered most of the system attributes in the definition matrix. Therefore, populating these attributes in the

definition matrix and calculating the resultant coupling metrics are more complicated and time consuming than any other coupling metric calculations.

One of the interconnection factors that couples procedure is the process of function call, where a variable in the body of the caller procedure is passed to the called procedure, which in turn is passed into another procedure and so on. Therefore the parameters of all called procedures appeared in the call path are aliases of that variable. Hence, each called procedure should have some value at the entry of procedure-variable in the definition matrix.

For the definition matrix to be accurate and complete, the aliases of parameters have to be resolved first. The algorithm proposed by Al-Nasser to populate the definition matrix does not resolve aliases problem.

Al-Nasser algorithm is modified for the following reasons:

1. To cater for parameter aliases problem.
2. To consider the number of times a global variable is accessed by a function.
3. There is no distinction between procedures and functions in C language, a procedure is a function that returns void (nothing). Therefore no need for a function to have an entry in the columns of the definition matrix, except when a function is called without any passed parameter .

To solve parameter aliases, special algorithm and routines are developed to build all parameter aliases for each function in the system. The built aliases are saved in the database for efficiency. However, it can be regenerated during run time.

Details of the algorithms and the tables are explained in section [4.3.3.1].

The original algorithm proposed by Al-Nasser considers any reference in the function body of global variable only once. In each function call if the variable is encountered then that variable is reconsidered and the entry in the definition matrix is incremented accordingly. In our opinion this does not reflect how strong the variable is bound to that function. Because the entry in definition matrix is incremented in each function call and incremented only once for all write, read and control operations.

The modified algorithm counts the number of times a variable is referenced inside the function body for read, write or control, and then increment the entry in the definition matrix accordingly. If the variable is referenced as a passed parameter to other function then it will be incremented during function call process. Local variables are considered and incremented only during function calls.

The modified algorithm assigns more accurate weight to each referenced global variable than the original algorithm.

A complete listing of the program is included in appendix C.

4.3.2.1 Main body of the algorithm

Figure 11 illustrates the modified algorithm in pseudo language.

The complete listing is included in appendix C.

```
For each function of the system under-consideration: /** loop 1 **/
        Add function name in the row of the definition matrix.
```

For each global variable used by this function and its attributes (read or write or control) has non zero value:

If global variable DOES NOT exists in any column of the definition matrix then

Add global variable as a new column in the definition matrix.

Get the type of this variable i.e., scalar or structure .

Calculate the number of times the variable used for read , write and control operation.

Update the entry in the definition matrix for current function and for current global variable.

End.

For each function-call done by this function order-by call sequence: /**loop 3 **/

Get the function-id of the called function.

For each passed-argument of this function-call order-by arg-position:

If the type of passed-arg is a variable or a field then do:

Determine the scope of the variable., i.e., local or global.

If variable DOES NOT exists in any column of the definition matrix then

Add variable as new column in the definition matrix.

Calculate the value to be added to entry of the definition matrix.

Update definition matrix entry for the current function and for the current variable.

Run a procedure that updates the definition matrix entries for all aliases of this variable including the called-function.

End.

If the type of passed-arg is an expression or a constant then do:

Find the parameter-name of the called function that matches the position of passed-argument.

If parameter name DOES NOT exists in any column of the definition then

Add parameter-name as new column in the definition matrix.

Calculate the value to be added to entry of the definition matrix.

Update the definition matrix entry for the current function and for the current parameter-name.

Run a procedure that updates the definition matrix entries for all aliases of this parameter-name include the called-function.

End.
End of each passed-argument.
End of each function-call.
End of populating the definition matrix.
Run a procedure that calculates the coupling matrix using the same method described in chapter 2. Section [2.4].
End of main body.

Figure 11. Algorithm that calculates CC coupling metrics.

<u>Procedure update-all-alias:</u>

Find parameter[P] under consideration.
For each parm-alias of parameter[P] do:
Find function[C] where function.fun-id= parm-alias.called-id.
Find parameter[P1] of function where parameter[P1].position = parm-alias.position.
determine the role of parameter[p1] in the called function, (i.e., control or data).
If function[C] has no row entry in the definition matrix then
Create a new row for function[C] in the definition matrix.
Update entry for function[C] and paramter[P] in the definition matrix.
End.
End procedure.

Figure 12. Procedure that updates all aliases of CC Matrix .

4.3.3 Resolving Aliases

Based on InterProcedural Alias Analysis (IPA) mentioned in appendix-A, a new algorithm is developed to uncover all parameter-to-parameter aliases.

The new algorithm is not completely based on IPA, it adopts the notation of worklist. The new algorithm is developed to work upon a system described in relational model proposed in this thesis.

Requirements:

Uncover all formal-to-formal aliases. This means find all parameters that are aliases of each parameter.

The aliases will be stored in the database. The generation of these aliases is performed only once after loading the system constructs into the database. Figure 13 illustrates the formal-to-formal parameter aliases.

```
Program X;
global variable g1,g2,g3;
call p1(g1,g2,g3);
Procedure P1(f1,f2,f3){
    call p2(f1,f2,f3);
    call p3(f3);
    call p4(f1,f1);
}
Procedure P2(f4,f5,f6){
    call p3(f4);
    call p4(f5,f6);
}
Procedure P3(f7){
    ....
}
Procedure P4(f8, f9){
    ....
}
```

Figure 13. Example of Parameters Aliases.

We need to find all parameter aliases for f1, f2, ..., f9 illustrated in figure 13.

The parameter aliases of f1 are: {f4, f7, f8, f9}.

The parameter aliases of f2 are: {f5, f8}.

The parameter aliases of f3 are: {f6, f7, f9}.

The parameter aliases of f4 are: {f7}.

The parameter aliases of f5 are: {f8}.

The parameter aliases of f6 are: {f9}.

The parameter aliases of f7 are: {}.

The parameter aliases of f8 are: {}.

The parameter aliases of f9 are: {}.

4.3.3.1 Uncovering aliases algorithm

Figures 14 and 15 respectively illustrate in pseudo language the algorithm to uncover parameter-to-parameter aliases. The complete listing is included in appendix C.

The worklist table used in the algorithm is explained below:

Table worklist.

Field caller-id /* this represents the function id that includes the parameter under consideration. */

Field caller-position /* this represents the position of the parameter in the parameter list of function whose fun-id is caller-id **/

Field called-id /* this represents the function id that holds a parameter which is an alias of the parameter under-consideration */.

Field called-position /* this represents the position of the parameter in the parameter list. The parameter is an alias of the parameter under consideration **/

The return-list table has the same fields as worklist table.

The main body of the algorithm

```
For each function [N] do: (i.e., caller)
        /*** the following discovers parameter aliases of at first level only,
                where the relation between two functions (caller & called) is
        considered **/

        For each parameter [P] of function[N] do:
                For each function-call made by this function do:
                        For each passed-argument of current function-call and argument-
                        name is [P] do:
                                Find worklist where caller = function [N]
                                and caller.position = parameter[P].position and
                                called-id= called-fun-id  and
                                called-position = passed-argument. position.
                                If worklist NOT found create worklist. Do:
                                        Worklist.caller= function[N].
                                        Worklist.caller-position= parameter[P].poistion.
                                        Worklist.called-id= called-fun-id.
                                        Worklist.called-position= passed-argument.
                                        position
                                End.
                        End.
                        /** Repetitively run a function that return a list of first level
                        parameter aliases of the called part of the worklist, if items in the
                        list do not exist then append them to worklist **/

                        Repeat for each worklist where worklist  processed =NO do:
                                Get a list of first_level_aliases for worklist.called-id and
                                worklist.called-postion.
                                For each item in the return-list do:
                                        if the item is not in the worklist then
                                                append item to the worklist.
                                End.
                                Current worklist. Processed=YES.
                        End.
                        For each worklist :
                                Save worklist in parm-alias table  making
```

```
                    Parm-alias.caller-id = function[N] fun-id.
                    Parm-alias.caller-position= parameter[P].position.
                    Parm-alias.called-id        = worklist.called-id.
                    Parm-alias.called-position  = worklist.called-position.
               End.
          End.
End.
```

Figure 14. Algorithm used to uncovering parameter aliases.

<u>Function returns first level aliases</u>

```
    Find called function [C]:
    FIND parameter of function where parameter-position = worklist.position .
    FOR EACH function-call made by function[C] do:
              FOR EACH passed-argument of the function-call and
              passed-argument.name = parameter name then DO:
                   /* it item not in the list then adds it to list */
                   FIND FIRST return-list WHERE return-list.caller-id=
                   function[C].fun-id AND
                   return-list.caller-pos= parameter.position and return-
                   list.called-id= called-fun-id AND return-list.called-pos=
                   passed-argument.position.
                   IF NOT AVAILABLE return-list THEN DO:
                      CREATE return-list.
                      return-list.caller-id=function[C].fun-id.
                      return-list.caller-pos= parameter.position.
                      return-list.called-id= called-fun-id.
                      return-list.called-pos= passed-argument.position.
                   END.
               END.
         Return return-list.
    End of procedure.
```

Figure 15. Procedure used to uncover first level aliases.

The final worklist of the example in Figure 13 is shown in table 18 below.

Caller-id	Caller-position	Called-id	Called-position
P1	F1	P2	F4

P1	F1	P3	F7
P1	F1	P4	F8
P1	F1	P4	F9
P1	F2	P2	F5
P1	F2	P4	F8
P1	F3	P2	F6
P1	F3	P3	F7
P1	F3	P4	F9
P2	F4	P3	F7
P2	F5	P4	F8
P2	F6	P4	F9

Table 18. List of resolved parameter aliases.

4.4 Implementation of Matrices

The 4GL language of the database does not support dynamic allocation of memory, which is needed to populate the definition and coupling metrics for cover-coefficient method. A new representation based on temp-tables is used to mimic the two-dimensional array. Three tables are created. The first represents the rows of the D-matrix. The second table represents the columns of the D-matrix, finally a third table is used to represent the entry of the matrix i.e., Cij.

The fields of the first table

Field j-num int this represents the j-entry.

Field j-name char name of the variable/parameter etc.

The fields of the second table

Field i-num int this represents the i-entry.

Field i-name char name of the name of the function.

The fields of the third table

Field j-num int this represents a pointer the j-column.

Field i-num int this represents a pointer to I-row.

Field value dec this represents the value at Cij.

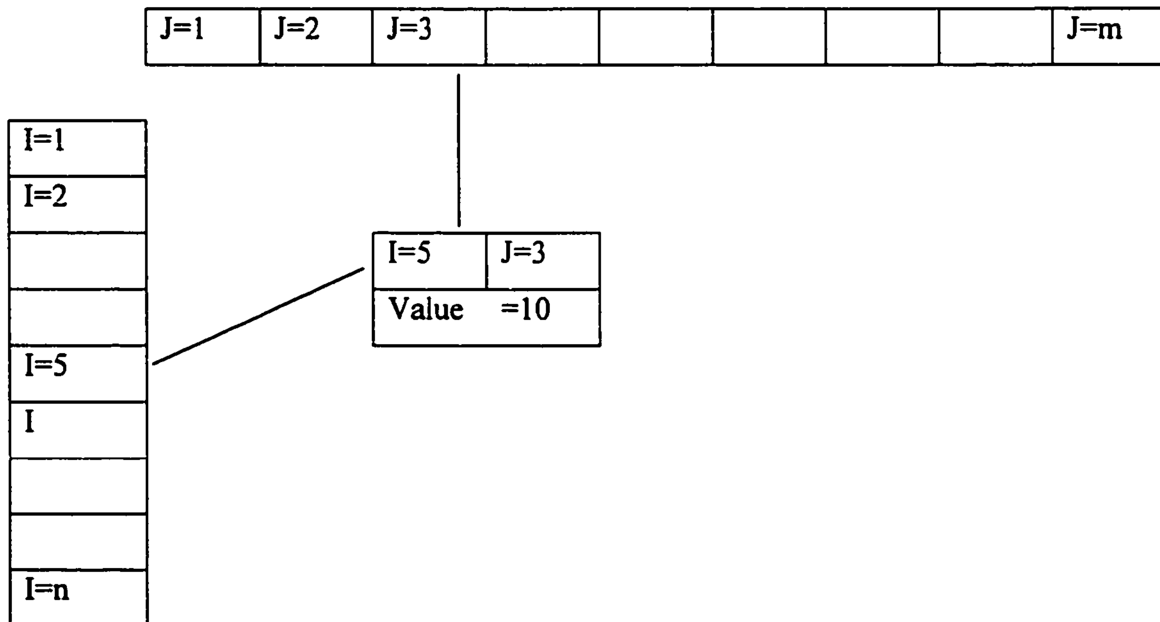


Figure 16. Matrix representation in relational temp tables.

Chapter 5

Study Cases

The system is applied against its own source code, i.e. the source code of the FCMS, and uploaded into the database.

Cover coefficient method and Dhama's method program were executed against the source code of the FCMS. The system was able to output the coupling of each function as well as the system coupling. The output of cover-coefficient method was saved in excel sheet for better visualization.

Three pairs of other small programs are also parsed and uploaded into the database. The functionality of these small programs is the same (sorting an array of integers), However each one has different implementation. Each pair of programs has the same implementation except one of its programs has extra redundant function.

5.1 Cover Coefficient method

5.1.1 Case 1

Cover coefficient method was applied against the source code that contains the functions of the FCMS. The definition and coupling metrics produced are included in appendix G. They are not included in this section because of their sizes.

Due to large size of the definition matrix, part of it was examined manually for global and parameter aliases. It was found to function as expected.

The produced coupling matrix was examined and found to contain many functions with zero couplings.

The source code is re-scanned to investigate such output. It was found that these functions with zero coupling are either:

Debugging functions used during the development phase of the parser such as print_simple_data.

Functions called by other modules in the system but included in the module under-consideration i.e., all functions of the FCMS are included in single file.

The behavior of having zero coupling suggests that functions can be clustered according to their coupling strengths. For example, functions with zero couplings in module A, can be included in a module B where they will be called.

5.1.2 Case 2

The Cover-Coefficient method was applied against three small pairs of programs called sort1,sort11, sort2, sort21, sort3, and sort31. The listings of the sorting programs are included in appendix D.

The functionality of all the six programs is to sort an array of integers. However, the second program of each pair has one extra redundant function. This is to test the sensitivity of the metrics when redundant functions are introduced.

A modified Cover-Coefficient method algorithm was also developed to produce the

definition matrix of the programs above. It is similar to the origianal one except that, it counts any global variables encounted inside any procedure once only, regradless of how many times the variable has been accessed.

Sort 1:

The main function uses a global variable vec (array of int) then passes it as a parameter to function sort. Function sort is responsible for actual sorting. The main function then prints the empty string. The definition and coupling matrices are shown in tables 19 and 20 respectively. Table 21 is the system and the average coupling of sort1.

	Main	sort	sort
	GV	LV	LV
	vec	k	k1
	SIMPLE	SIMPLE	SIMPLE
Main	10	0	0
Sort	3	1	1
Swap	0	1	1

Table 19. CC definition matrix for program sort 1.

	Main	sort	swap
Main	0.769231	0.230769	0
Sort	0.461538	0.338462	0.2
Swap	0	0.5	0.5

coupling
0.231
0.662
0.5

Table 20. CC coupling matrix for program sort 1.

SYSTEM COUPLING	1.392308
AVERAGE COUPLING	0.464103

Table 21. CC system and average coupling of program sort 1.

The definition and the coupling matrices of the second program in the pair are shown in Table 22 and Table 23 respectively.

	Main	Sort	Sort
	GV	LV	LV
	Vec	K	K1
	SIMPLE	SIMPLE	SIMPLE
is_greater	0	0	0
Main	10	0	0
Sort	3	1	1
Swap	0	1	1

Table 22. CC definition matrix of sort 11.

	Is_greater	main	Sort	swap
is_greater	0	0	0	0
Main	0	0.769231	0.23076	0
Sort	0	0.461538	0.33846	0.2
Swap	0	0	0.5	0.5

coupling
0
0.231
0.662
0.5

Table 23. CC coupling matrix of sort 11.

SYSTEM COUPLING	1.3923
AVERAGE COUPLING	0.34825

Table 24. CC system and average coupling of sort11.

The definition and the coupling matrices of sort 1 produced by the modified algorithm are shown in Table 25 and Table 26 respectively.

	Main	sort	Sort
	GV	LV	LV
	Vec	k	k1
	SIMPLE	SIMPLE	SIMPLE
Main	5	0	0
Sort	3	1	1
Swap	0	1	1

Table 25. CC definition matrix of program sort 1. (modified alogrithm).

	main	sort	Swap
Main	0.625	0.375	0

Sort	0.375	0.425	0.2
Swap	0	0.5	0.5

Table 26. CC coupling matrix of program sort 1. (modified algorithm).

SYSTEM COUPLING	1.45
AVERAGE COUPLING	0.483333

Table 27. CC system and average coupling of program sort 1 (modified algorithm).

Sort 2:

The main function uses a local variable vec (array of int) then passes it as a parameter to function sort. Function sort is responsible for actual sorting.

The main function then prints and empty string.

	main	sort	sort
	LV	LV	LV
	vec	k	k1
	SIMPLE	SIMPLE	SIMPLE
Main	1	0	0
Sort	3	1	1
Swap	0	1	1

Table 28. CC definition matrix of sort 2.

	main	Sort	swap
Main	0.25	0.75	0
Sort	0.15	0.65	0.2
Swap	0	0.5	0.5

coupling
0.75
0.35
0.5

Table 29. CC coupling matrix of sort 2.

SYSTEM COUPLING	1.6
AVERAGE COUPLING	0.533333

Table 30. CC system and average coupling of sort 2.

The definition and the coupling matrices of the second program in the pair are shown in Table 31 and Table 32 respectively.

	Main	sort	sort
	LV	LV	LV
	Vec	k	k1
	SIMPLE	SIMPLE	SIMPLE
Is_greater	0	0	0
Main	1	0	0
Sort	3	1	1
Swap	0	1	1

Table 31. CC definition matrix of sort 21.

	Is_greater	main	sort	Swap
is_greater	0	0	0	0
Main	0	0.25	0.75	0
Sort	0	0.15	0.65	0.2
Swap	0	0	0.5	0.5

coupling
0
0.75
0.35
0.5

Table 32. CC coupling matrix of sort 21.

SYSTEM COUPLING	1.6
AVERAGE COUPLING	0.4

Table 33. CC system and average coupling of sort 21.

The definition and the coupling matrices of sort 2 produced by the modified algorithm are shown in Table 34 and Table 35 respectively.

	Main	sort	sort
	LV	LV	LV
	Vec	k	k1
	SIMPLE	SIMPLE	SIMPLE
main	1	0	0
sort	3	1	1
swap	0	1	1

Table 34. CC definition matrix of program sort2. (modified algorithm).

	main	sort	swap
main	0.25	0.75	0
sort	0.15	0.65	0.2
swap	0	0.5	0.5

coupling
0.75
0.35
0.5

Table 35. CC coupling matrix of program sort 2.(modified algorithm).

SYSTEM COUPLING	1.6
AVERAGE COUPLING	0.533333

Table 36. CC system and average coupling of sort 2. (modified algorithm).

Sort 3:

The main function uses a global variable vec (array of int) then calls a function sort. Function sorts the global vec. Sort is responsible for actual sorting and has no parameters.

The main function then prints an empty string.

	main	sort	Sort	Sort
	GV	FN	LV	LV
	vec	sort	K	k1
	SIMPLE	SIMPLE	SIMPLE	SIMPLE
Main	5	5	0	0
Sort	34	5	1	1
Swap	0	0	1	1

Table 37. CC definition matrix of sort 3.

	main	sort	Swap
Main	0.314103	0.685897	0
Sort	0.167292	0.808318	0.02439
Swap	0	0.5	0.5

coupling
0.686
0.191
0.5

Table 38. CC coupling matrix of sort 3.

SYSTEM COUPLING	1.37758
AVERAGE COUPLING	0.459193

Table 39. CC system and average coupling of sort 3.

The definition and coupling matrices of the second program in the pair are shown Table 40 and Table 41 respectively.

	main	sort	Sort	sort
	GV	FN	LV	LV
	vec	sort	K	k1
	SIMPLE	SIMPLE	SIMPLE	SIMPLE
is_greater	0	0	0	0
Main	5	5	0	0
Sort	34	5	1	1
Swap	0	0	1	1

Table 40. CC definition matrix of sort 31.

	is_greater	main	Sort	swap
is_greater	0	0	0	0
Main	0	0.314103	0.685897	0
Sort	0	0.167292	0.808318	0.02439
Swap	0	0	0.5	0.5

Coupling
0
0.686
0.191
0.5

Table 41. CC coupling matrix of sort 31.

SYSTEM COUPLING	1.37758
AVERAGE COUPLING	0.344395

Table 42. CC system and average coupling of sort 31.

The definition and the coupling matrices of sort 3 produced by the modified algorithm are shown in Table 43 and Table 44 respectively.

	Main	sort	Sort	sort
	GV	FN	LV	LV
	Vec	sort	K	k1
	SIMPLE	SIMPLE	SIMPLE	SIMPLE
Main	5	5	0	0
Sort	7	5	1	1
Swap	0	0	1	1

Table 43. CC definition matrix of program sort 3 (modified algorithm).

	Main	sort	Swap
Main	0.458333	0.541667	0
Sort	0.386905	0.541667	0.071429
Swap	0	0.5	0.5

coupling
0.542
0.458
0.5

Table 44. CC coupling matrix of program sort 3 (modified algorithm).

SYSTEM COUPLING	1.5
AVERAGE COUPLING	0.5

Table 45. CC system and average coupling of program sort 3 (modified algorithm).

The system couplings for the three pair of sorting programs do not agree with the intuition that access of global variables should have stronger coupling with the exception of modified program in the case of sort 1 and sort 3. The program sort 2 which has no global variable, shows highest coupling value. This unexpected behaviour of the coupling method could be due to the small size of the programs used, or the weighting scheme might need to be further refined. The system coupling of cover coefficient method shows insensitivity of redundant function, which makes the method more suitable to program clustering. However, the average coupling is affected by the number of redundant functions.

5.2 Dhama's method

5.2.1 Case 1

Dhama's method was applied against the source code that contains the FCMS.

The output obtained is shown in Table 46.

Process_calls_parm	00.50
Process_one_call_parm	00.1429
Process_single_parm	00.0417
return_name	00.1111

Return_simple_type	00.125
add_type	00.0455
Process_array_pointer	00.0667
Add_complex_type	00.0714
Return_type_name	00.1111
Initialize_data_types	01.00
Add_simple_type	00.0588
Return_type_num	00.50
Print_simple_type	00.50
Print_complex_type	00.00
make_var	00.10
make_field	00.3333
add_var_to_list	00.0769
Add_field_to_list	00.1667
print_var_list	00.1429
is_complex	01.00
brace_name	00.20
make_fun_call	00.50
add_call_to_list	00.20
print_fun_calls	00.1429
Make_parameter_node	00.25
Add_para_to_list	00.1667
Print_parameter_list	00.50
filter_comp	00.25
is_expression	00.3333
Is_single_operand	01.00
get_exp_var_list	00.1429
Add_str_to_exp_var	00.50
get_code_name	01.00
process_enum	01.00
var_exist	00.20
modify_var	00.0769
Process_cond_exp	00.1429
Process_read_write_exp	00.0667
get_exp_var1	00.037
extend_var_list	00.0556
Get_component_name	00.1111
break_comp	00.0588
mk_parm	00.1667
dump_data	00.0286
quote_str	00.0435
Get_system_name	00.50
get_last_call_id	01.00
Write_last_call_id	00.50
single_fun_id	00.50

get_type_id	00.3333

Table 46. Dhama's system coupling of FCMS.

***** System Coupling **** 0.30**

Lower number indicates stronger coupling.

5.2.2 Case 2

Dhama's method was applied against the same set of sorting programs mentioned in previous section.

Sort 1:

The system coupling obtained from Dhama's method when applied against sort 1 is shown in Table 47.

Swap	0.200
Sort	0.167
Main	0.333
System coupling	**0.233**

Table 47. Dhama's system coupling of sort 1.

The system coupling of second program in the pair is shown in Table 48:

Is_greater	0.250
Swap	0.200
Sort	0.167
Main	0.333
System coupling	**0.238**

Table 48. Dhama's System coupling for sort 11.

Sort 2:

The system coupling obtained from Dhama's method when applied against sort 2 is shown in Table 49.

Swap	0.200
Sort	0.167
Main	0.500
System coupling	**0.289**

Table 49. Dhama's system coupling for sort 2.

The system coupling of second program in the pair is shown in Table 50.

Is_greater	0.250
Swap	0.200
Sort	0.167
Main	0.500
System coupling	**0.279**

Table 50. Dhama's system coupling for sort 21.

Sort 3:

The system coupling obtained from Dhama's method when applied against sort 3 is shown in Table 51.

Swap	0.200
Sort	0.200
Main	0.333
System coupling	**0.244**

Table 51. Dhama's system coupling for sort 3.

The system coupling of second program in the pair is shown in Table 52.

Is_greater	0.250
Swap	0.200
Sort	0.200
Main	0.333
System coupling	**0.245**

Table 52. Dhamas system coupling for sort 31.

The outputs of Dhama's method do not agree fully with the intuition that access of global variables should have stronger coupling, i.e. sort 3 has better coupling value than sort 1 even though the function *sort* in program sort 3 access global variable vec. The reason for such output is that, in the case of program sort-1 , the data and control parameters coupling of function *sort* has higher value than in program sort-3 which is zero, and contribution of global variable coupling in sort-3 is less than the contribution of data and control parameter coupling in sort-1 to the overall coupling. The other reason could the size of programs tested in this chapter are too small. The usage of local variables like in program sort-2 have good effect in de-coupling of the system. Dhama's method is not sensitive enough to distinguish whether a function belongs to the module or not. Example, function is_greater has never been used but it contributes to global system coupling.

Chapter 6

Conclusion

Coupling is very important factor in determining the quality of a software system. Many coupling metrics are not subjected to production programs due to the difficulties in extracting required attributes and the unavailability of tools that achieve this task automatically. In this thesis, a general model to measure the coupling was presented. Automatic tools to extract and store system attributes were developed. The FCMS allowed us to test coupling methods that have not been subjected to large production programs before. For example, Cover-Coefficient and Dhama's methods were automatically obtained for the source code of the FCMS.

6.1 Contribution

This thesis has contributed to the research the followings:

- It has provided a general framework to measure coupling metrics. The framework acts as standard platform for many coupling metrics calculations surveyed in the literature.
- It has leveraged the constructs of imperative languages into a higher layer of abstraction. This abstraction maintains the necessary details needed to achieve

metrics calculations while hiding the specifics of the languages or the environment.

- It has provided base for fair and unbiased comparison and evaluation of different coupling metrics calculation. This is because all metrics calculations will be subjected to the same environment and same set of data.
- The abstract format can be adopted by many analysis schemes other than coupling metrics. For example, it can be used as basis for clustering algorithms.
- The framework based on abstraction developed in this thesis is capable of providing the following system characteristics and properties:

⇒ Functions dependencies.

⇒ Fan In(s) and Fan Out(s) of functions.

⇒ Number of Lines of Source Code.

⇒ Number of functions, global and local variables.

⇒ Checks for unused variables.

⇒ Checks for variable aliases, i.e. appearance of the same name in different modules.

- Based on InterProcedural Alias Analysis (IPA) an algorithm that depends on the abstract format was developed to uncover variable-to-formal and formal-to-formal aliases.
- A complete system was developed to illustrate the applicability of the framework. It has the parsing utilities; the storing facilities and programs needed to calculate coupling metrics. It has achieved the metrics assessment requirement, i.e. automatically *collecting, storing and analyzing* software.

6.2 Future Works

This thesis can be further extended to achieve the followings:

- A complete general coupling measurement system for high level languages can be achieved by extending further the abstract format proposed in this thesis, to include all the attributes and the factors of object oriented paradigm or any other paradigm.
- The abstract format proposed for imperative languages can be extended to cater for nested functions.Local variables of the same name defined in nested blocks inside a single function can be handled a more accurate way.
- Variable-to-variable aliases can be incorporated into the model.
- The model can be extended to capture extra system attributes required by different metric schemes, such as:
  - Variables dependencies.
  - Number of loops.
  - Conditional branches.
  - Other system attributes.
- There are many characteristics and properties of software systems that can be further analyzed. For example, the framework presented in this thesis can be used to develop various clustering programs. The clustering programs may be applied to cluster software systems. Repetitively clustering then calculating coupling of software programs, it is possible to find the best correlation between a pair of clustering algorithm and coupling metric method.

Appendix A

The following illustrate the algorithm proposed by [Maye93] to uncover parameter-to-parameter aliases.

Referencing using parameters:

Based on a method called InterProcedural Alias Analysis (IPA) proposed by [Maye93], parameter passing references can be uncovered and identified in program. There are two different types of calls with formal reference parameters, give rise to aliasing [Maye93].

1. Global-to-formal aliasing is caused by passing an object g (global variable) as an actual parameter.
2. Formal-to-formal aliasing is caused by passing some variable(s) v two or more times in the same call as an actual parameters, for example callee(v,v).

Each type above has different detection algorithm referred as phase one and phase two.

Phase one algorithm:

The purpose of phase 1 is to uncover all type 1 alias relations; these exist between globals and formals. A binding graph B is the primary data structure to represent these relations for formals.

Each node in β represents a unique formal parameter f_i. A vertex is inserted into β if either a global g or a formal reference parameter $f1$ is passed in a call as an

actual parameter and bound to respective formal reference parameter $f1$ of the callee. In the former case, passing a global g, a singleton node is created in β for the formal $f2$ that g is bound to and g is added to the alias set of $f2$. In the latter case, up to two nodes are inserted, one for $f1$ and the other for $f2$. An edge $f1$ -> $f2$ is also established.

Each f_i in β is attributed with its alias set. Any directed edge $f_i \to f_j$ expresses that formal f_i is used as an actual in the call and bound to formal f_j.

Using Tarjan's algorithm to find maximal strongly connected components (SCC), which in used against reversed binding graph; SCCs will be identified and replaced by a representative node. Taking the advantage of Tarjan's algorithm when used against reversed binding graph, a data flow representation of nodes can be constructed yielding identified aliases of type 1.

Example 1

Program X;

global variable g1,g2,g3;

call p1(g1,g2,g3);

Procedure *P1*(f1,f2,f3){

call p2(f1,f2,f3);

}

Procedure *P2*(f4,f5,f6){

call p1(g3,f4,f5);

```
        call p3(f5,f6);

}

Procedure P3(f7,f8){

        ....

}
```

Figure 1. Outline listing of example 1 for IPA method.

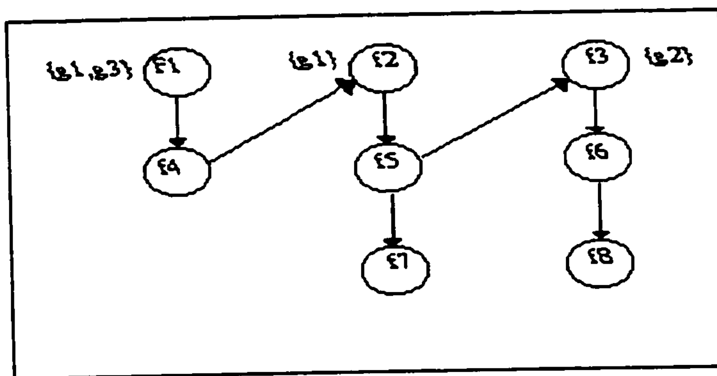


Figure 2. Alias set for formals of example 1.

f1	f2	F3	f4	f5	f6	f7	f8
{g1,g3 }	{g1,g3}	{g1,g2, g3}	{g1,g3}	{g1,g3}	{g1,g2, g3}	{g1,g3}	{g1,g2, g3}

Figure 3. Aliases set for formals.

<u>Phase two algorithm:</u>

The purpose of phase 2 is to uncover all type 2 alias relation: these can exist between any two formals $f1$ and $f2$ of the same procedure p.

The method is based on primary data structure called work list Ω . Each element of

Ω contains triple of the two parameter aliases and their callee's name.

The method works as follow:

1. Scan all calls in the program for calls that have objects used more than once as an actual parameter. For example callee(v,v) to build initial Ω .On finding such call , probe the parameter of the callee (f1,f2) and insert , callee name, f1 and f2 into Ω .
2. Scan all calls that appear in Ω which uses f1 and f2 as actual parameter, if found insert into Ω in the same way as step 1 until no more element in Ω need to be processed
3. Form a union of all parameters in work list Ω of each others, for example for each pair(f_i , f_j) in Ω : alias(f_i)alias(f_i) $\cup$ { f_j } and alias(f_j)alias(f_j) $\cup$ { f_i }.

The union set formed in step 3 is all the aliases of type 2 in the program.

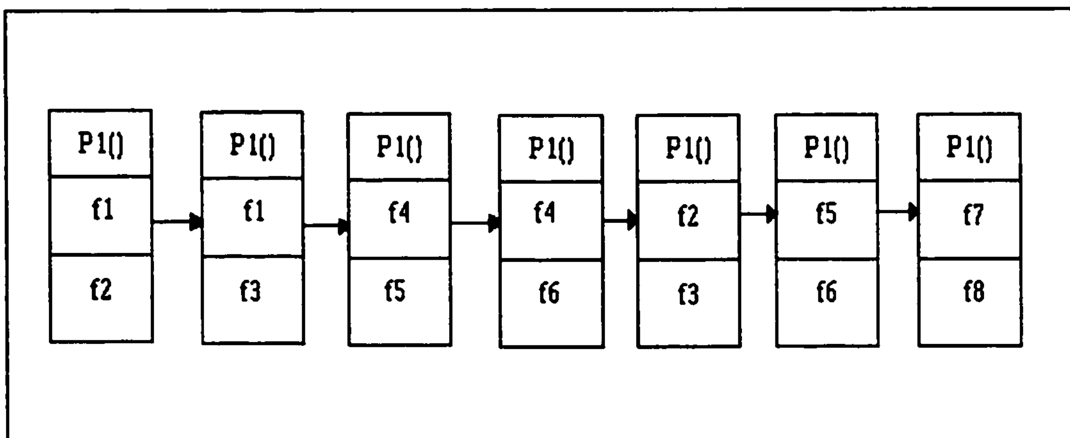


Figure 4. The work list Ω for example 1 of IPA.

Alias relations for formals of example 1

f1	f2	f3	f4	f5	f6	F7	f8
{f2,f3}	{f1,f3}	{f1,f2}	{f5,f6}	{f4,f6}	{f4,f5}	{f8}	{f8}

Figure 5. Alias relations for formals of example 1.

Appendix B

The following is the listing of the source code of the extended functions of the general coupling measurement system.

Figure 1. Listing of the extended funciton of the system.

```
/*** this is the include-file holding the constants and some of the function prototypes ***/
#define MAX_NUM_OF_FUN  200
#define FUNC_ARRAY function_node fun_vec[MAX_NUM_OF_FUN];

#define NAME_LENGTH 60
#define TYPE_LENGTH 100
#define MAX_VAR_NUM 300
#define STRUCT_STR "structure"
#define UNION_STR  "union"
struct var_template {
        char     name[NAME_LENGTH];
        unsigned         data_type_id;
        int   control;
        int       read;
        int   write;
        struct var_template * next;
};
typedef struct var_template var;
typedef var * VAR_LIST;

struct passed_parameters {
                char            parameter_type[TYPE_LENGTH];
                char            parameter_name [NAME_LENGTH];
                char            exp_vars [TYPE_LENGTH];
                struct passed_parameters * next_para;
                };
typedef struct passed_parameters PARAMETERS;
typedef PARAMETERS  * PARA_LIST ;

struct function_call {
        PARA_LIST               passed_args;
        char            called_name[NAME_LENGTH];
        struct function_call *next;
        };
typedef struct function_call SINGLE_FUN_CALL;
typedef SINGLE_FUN_CALL  *CALL_LIST;

struct field_data_type {
                char field_name[NAME_LENGTH];
                unsigned  data_type_id;
                struct field_data_type * next;
};

typedef struct field_data_type FIELD_DATA;
typedef FIELD_DATA *FIELD_LIST;

struct simple_data_type{
```

```
                char data_type_name[TYPE_LENGTH];
                int def_module;
                char module_name[NAME_LENGTH];
                unsigned   data_type_id;
                int is_pointer;
                int  type_link_num;
                FIELD_LIST fields ;
        };

typedef struct simple_data_type SIMPLE_DATA ;

struct fun_def {
        char name[NAME_LENGTH];
        unsigned   return_data_type_id;
        VAR_LIST   local_var;
        VAR_LIST   global_var_used;
        VAR_LIST   para_list;
        CALL_LIST  fun_calls;
        int            no_of_stmt;

};

typedef struct fun_def function_node;
char * return_type(tree);
char * return_name(tree);
VAR_LIST make_var(char *, unsigned   );
CALL_LIST make_fun_call(char *, PARA_LIST);
tree add_new_node(tree, tree);
int add_var_to_list(VAR_LIST,VAR_LIST);
int add_call_to_list(CALL_LIST, CALL_LIST);
void process_calls_parm(tree);
void print_parameter_list(PARA_LIST);
char * get_code_name(int);
char * return_type_name(unsigned );
int add_type(tree,char*);
void make_add_var(char *, unsigned , VAR_LIST);
enum access_type { WRITE, READ, CONTROL};
typedef enum access_type ACCESS_TYPE;
int modify_var(char*,int,ACCESS_TYPE, unsigned);
int process_cond_exp(tree, int);
void process_pending_exp(tree, int);
int process_read_write_exp(tree, int,ACCESS_TYPE);
void get_component_name(tree, char*,char*);
#define MY_CODE(x)        get_code_name(TREE_CODE(x))



/*** The following is the listing of the source code of the GCSM ***/

#define COMPLEX_T_N(x)  (TREE_CODE(x)==UNION_TYPE || TREE_CODE(x)==RECORD_TYPE)
#define  IS_ENUM(x)   (TREE_CODE(x)==ENUMERAL_TYPE)
#define ARRAY_POINTER(x)   (TREE_CODE(x)==POINTER_TYPE || TREE_CODE(x)==ARRAY_TYPE)
#define IS_SIMPLE1(x)   (TREE_CODE(x)==INTEGER_TYPE || TREE_CODE(x)==VOID_TYPE)
#define IS_SIMPLE2(x)   (TREE_CODE(x)==CHAR_TYPE || TREE_CODE(x)==REAL_TYPE)
#define IS_SIMPLE(x)   (IS_SIMPLE1(x) || IS_SIMPLE2(x))

#define mydbg(x)    printf(x);

#undef DEFTREECODE
#define DEFTREECODE(SYM, NAME, TYPE, LENGTH) NAME,

char *string_tree_code_type[] = {
```

```
#include "tree.def"
};

#undef DEFTREECODE

#include "config.h"
#include "tree.h"
#include "flags.h"
#include "c-tree.h"
#include "c-lex.h"
#include <stdio.h>
#include "gcms.h"
int dummy_counter=0;
extern int tariq_start_fun;
extern int global_num_of_fun;
extern VAR_LIST pending_var;
extern stmt_count;
extern FUNC_ARRAY;
char  temp_para_type[TYPE_LENGTH];
char  temp_para_name[NAME_LENGTH];
char  temp_para_exp_var[TYPE_LENGTH];
char  last_type_found[TYPE_LENGTH];
char  str_name[NAME_LENGTH];
char  full_field_name[NAME_LENGTH * 4];
int last_cond_type;
VAR_LIST temp_var_list;
VAR_LIST global_var_list;
extern SIMPLE_DATA  s_data_type[MAX_VAR_NUM];
PARA_LIST rtn_para_list;
void process_single_parm(tree);
PARA_LIST make_parameter_node(char *, char * , char *);
char * brace_name(char*,char *, char *);
void process_one_call_parm(tree);
char * filter_comp(tree, char *);
char *  get_exp_var_list(tree,int);
void get_exp_var1(tree, int, int );
int process_array_pointer(tree,tree,char*);
int add_complex_type(tree, tree, unsigned );
FIELD_LIST make_field(char *, unsigned );
int process_enum(tree, tree);
void get_component_name(tree, char * , char*);
void break_comp(tree);
int quote_str(char*, char*);
#define TYPE_EXISTS(x)    ( strcmp(return_type_name(x),"")!=0)
char * single_fun_id(char *, char *);
unsigned get_type_id( char *);
/****************************************
This is a entry routine to process the parameters of a funciton call . This seems redundant however,
Resolve recursion problem.
****************************************/

void process_calls_parm(tree x) {
tree p;
        p=x;
        while(p){
                process_one_call_parm(p );
                p=TREE_CHAIN(p);
        };
};
/*******************************************
This routine trims off  type conversion of C lanaguage before processing
a parameter.
```

```
******************************************/
void process_one_call_parm(tree x)
{
tree p;
p=x;
        if(TREE_CODE(p)==NOP_EXPR || TREE_CODE(p)==CONVERT_EXPR)
                p=TREE_OPERAND(p,0);
        process_single_parm(p);
};
/******************************************
This routine process each single parameter individually and record
its name ,type  and purpose.
******************************************/
void  process_single_parm(tree x )
{
static  PARA_LIST tmp_para;
tree i;
tree t;
        i=copy_node(x);
        strcpy(temp_para_exp_var,"");
        if (TREE_CODE(i)==VAR_DECL || TREE_CODE(i)==PARM_DECL ||
                TREE_CODE(i)== FUNCTION_DECL){
                if (TREE_CODE(i)==VAR_DECL)
                        strcpy(temp_para_type,"VARIABLE");
                if(TREE_CODE(i)==PARM_DECL)
                        strcpy(temp_para_type,"PARAMETER");
                if(TREE_CODE(i)==FUNCTION_DECL)
                        strcpy(temp_para_type,"FUNCTION_NAME");
                strcpy(temp_para_name,return_name(i));

                if (TREE_CODE(i)!=FUNCTION_DECL)
                        process_read_write_exp(i,global_num_of_fun,PASS_PARM);
        }
        else if(TREE_CODE(i)==TREE_LIST) {
                if(TREE_VALUE(i)!=NULL) {
                        process_one_call_parm(TREE_VALUE(i));
                        return;
                };
        }
        else if(TREE_CODE(i)==CALL_EXPR){
                strcpy(temp_para_name,return_name(TREE_OPERAND(TREE_VALUE(i),0)));
                strcpy(temp_para_type,"FUNCTION_CALL");
        }
        else if(TREE_CODE(i)==COMPONENT_REF ){
                strcpy(temp_para_exp_var,
                        filter_comp(i,&temp_para_name[0]) );
                strcpy(temp_para_type,"FIELD");
                process_read_write_exp(i,global_num_of_fun,PASS_PARM);
        }
        else if(TREE_CODE(i)==INDIRECT_REF) {
          process_one_call_parm(TREE_OPERAND(i,0));
                return;
        }
        else if(TREE_CODE(i)==ARRAY_REF ){
                process_one_call_parm(TREE_OPERAND(i,0));
                return;
        }
        else if(TREE_CODE(i)==ADDR_EXPR){
                strcpy(temp_para_name,return_name(TREE_OPERAND(i,0)));
                strcpy(temp_para_type,"VARIABLE");
                strcpy(temp_para_exp_var,"");
                process_read_write_exp(i,global_num_of_fun,PASS_PARM);
```

```
        }
        else if(TREE_CODE(i)==STRING_CST || TREE_CODE(i)==REAL_CST ||
                TREE_CODE(i)==INTEGER_CST ){
                strcpy(temp_para_type,"CONSTANT");
                strcpy(temp_para_name,"");
        }
        else if(TREE_CODE(i)==NOP_EXPR) {
                process_one_call_parm(TREE_OPERAND(i,0));
                return;
        }
        else if( is_expression(i)) {
                strcpy(temp_para_exp_var,"");
                strcpy(temp_para_name,"");
                strcpy(temp_para_type,"EXPRESSION");
                strcat(temp_para_exp_var,get_exp_var_list(i,1 ));
                strcpy(temp_para_name,"");
        }
        else {
                strcpy(temp_para_type,string_tree_code_type[TREE_CODE(i)]);
                strcpy(temp_para_name,"");
        };
        tmp_para=make_parameter_node(temp_para_type,temp_para_name,temp_para_exp_var);
                if(add_para_to_list(rtn_para_list,tmp_para)==0)
                        rtn_para_list= tmp_para;
}


/*****************************************
This routine simple returns the name of variables, functions, parameters etc.
As string of characters.
*****************************************/
char * return_name( tree l)
{

if( (TREE_CODE(l)==FUNCTION_DECL) ||
        (TREE_CODE(l)==PARM_DECL) ||
        (TREE_CODE(l)==VAR_DECL)  ||
        (TREE_CODE(l)==FIELD_DECL) ) {
        return (IDENTIFIER_POINTER(DECL_NAME(l)));

}

else  return("");
}

/*******************************************************************
This routine returns type of var in string format
*******************************************************************/
char * return_simple_type(tree l, tree origin)
{

tree p;
static char rtn_str[TYPE_LENGTH];

rtn_str[0]='\0';

if( (TREE_CODE(l)==VOID_TYPE)  ||
        (TREE_CODE(l)==CHAR_TYPE) ||
        (TREE_CODE(l)==INTEGER_TYPE) ||
        (TREE_CODE(l)==REAL_TYPE)       ){
        if(TREE_CODE(origin)==TYPE_DECL) {
```

```
                strcpy(rtn_str,IDENTIFIER_POINTER(DECL_NAME(origin)));
                return (rtn_str);
        }
        else
                return(string_tree_code_type[TREE_CODE(l)]);
}
else if(TREE_CODE(l)==POINTER_TYPE)
        return("ptr-to-");

else if(TREE_CODE(l)==ARRAY_TYPE) {
                return("array-of-");
}
else if(TREE_CODE(l)==RECORD_TYPE || TREE_CODE(l)==UNION_TYPE ||
TREE_CODE(l)==QUAL_UNION_TYPE){
        if(TREE_CODE(origin)==TYPE_DECL || TREE_CODE(origin)==VAR_DECL
                || TREE_CODE(origin)==PARM_DECL || TREE_CODE(origin)==FUNCTION_DECL){
                if(TYPE_NAME(l)==NULL){ /* it a typedef of var/parm with no-tag
                                use VAR-NAME as type */
                        strcpy(rtn_str,brace_name(IDENTIFIER_POINTER(DECL_NAME(origin)),"<",">"));

                }
                else if (TREE_CODE(TYPE_NAME(l))==IDENTIFIER_NODE) {
                        strcpy(rtn_str,brace_name(IDENTIFIER_POINTER(TYPE_NAME(l)),"<",">") );

                } else { /* it a type-of user-defined type i.e USER-TYPE *ptr ,
                        USER-TYPE must be already defined, hence get the name using TYPE_UID */

                        strcpy(rtn_str,return_type_name(TYPE_UID(l)));
                };
        }

        else strcpy(rtn_str,"");
        return(rtn_str);
}
else if(TREE_CODE(l)==ENUMERAL_TYPE){
        return ("enum");

}
else if(TREE_CODE(l)==FUNCTION_TYPE)
        return("function-[rtns]-");
else{
        strcpy(rtn_str,"");
        return rtn_str;
        }
}
/************************************************************/
int add_type(tree l, char *fn_name){

tree t;
char new_type_to_add[TYPE_LENGTH];
/*int new_type_to_add[TYPE_LENGTH];*/
unsigned type_num;

        strcpy(new_type_to_add,"");
        type_num=TYPE_UID(l);
/*DDD*/
        switch (TREE_CODE(l)) {

                case TREE_LIST: { /** this is used when adding tags **/
                        t=TREE_VALUE(l);
                        type_num=TYPE_UID(t);
                        if(COMPLEX_T_N(t)==0 ) {}
```

```
                if(TYPE_NAME(t)!=NULL){}
                break;
        }
        case TYPE_DECL: {  /** used when typedef is found **/
                t=TREE_TYPE(l);
                type_num=TYPE_UID(t);
                if(TYPE_EXISTS(type_num))
                        break;
                if(COMPLEX_T_N(t)==0) { /* simple_typedef */
                        if(ARRAY_POINTER(t)==NULL){
                        strcpy(new_type_to_add,
                                IDENTIFIER_POINTER(DECL_NAME(l)));

                                add_simple_type(new_type_to_add,type_num,0,fn_name);
                         } else
                                process_array_pointer(t,l, fn_name);
                        break;
                }
                if(TYPE_NAME(t)==NULL) {
                        strcpy(new_type_to_add,IDENTIFIER_POINTER(TYPE_NAME(l)));
                }
                else {

strcpy(new_type_to_add,IDENTIFIER_POINTER(DECL_NAME(TYPE_NAME(t))));
                }
                add_simple_type(new_type_to_add,type_num,0,fn_name);
                add_complex_type(t,l,type_num);

                break;
        }
        case VAR_DECL:
        case PARM_DECL: {
                t=TREE_TYPE(l);
                type_num=TYPE_UID(t);

                if(TYPE_EXISTS(type_num))
                        break;
                else if(COMPLEX_T_N(t)==0 ) {

                        if(ARRAY_POINTER(t))
                                process_array_pointer(t,l, fn_name);
                        else if(IS_ENUM(t)) {
                                if(TYPE_NAME(t)!=NULL)
                                        strcpy(new_type_to_add,
                                        IDENTIFIER_POINTER(TYPE_NAME(t)));
                                else

strcpy(new_type_to_add,IDENTIFIER_POINTER(DECL_NAME(l)));

                                add_simple_type(new_type_to_add,type_num,0,fn_name);
                        }
                        else {
                                strcpy(new_type_to_add,
                                IDENTIFIER_POINTER(DECL_NAME(TYPE_NAME(t))));
                                add_simple_type(new_type_to_add,
                                        type_num,0,fn_name);
                        };
                        break;
                }
                if(TYPE_NAME(t)==NULL) { /* make_var_name as type-name */
                        strcpy(new_type_to_add,IDENTIFIER_POINTER(DECL_NAME(l)));
                }
```

```
                    else if(TREE_CODE(TYPE_NAME(t))==IDENTIFIER_NODE){
                                strcpy(new_type_to_add,
                                        IDENTIFIER_POINTER(TYPE_NAME(t)));
                    }
                    else if(TREE_CODE(TYPE_NAME(t))==TYPE_DECL) {
                                /* ignore because it will be allready added
                                */
                    };
                    add_simple_type(new_type_to_add,type_num,0,fn_name);
                    add_complex_type(t,l, type_num );
                    break;
            }
            case FUNCTION_DECL: {
                    t=TREE_TYPE(TREE_TYPE(l)); /*tree_type(fun_type)*/
                    type_num=TYPE_UID(t);
                    if(TYPE_EXISTS(type_num))
                            break;
                    if(COMPLEX_T_N(t)==0){

                            if(ARRAY_POINTER(t))
                                    process_array_pointer(t,l,fn_name);
                            else if(IS_ENUM(t)) {
                                    if(TYPE_NAME(t)!=NULL)
                                            strcpy(new_type_to_add,
                                            IDENTIFIER_POINTER(TYPE_NAME(t)));
                                    else

    strcpy(new_type_to_add,IDENTIFIER_POINTER(DECL_NAME(l)));

                                    add_simple_type(new_type_to_add,type_num,0,fn_name);
                            }
                            else {
                                    strcpy(new_type_to_add,
                                    IDENTIFIER_POINTER(DECL_NAME(TYPE_NAME(t))));
                                    add_simple_type(new_type_to_add,type_num,0,fn_name);
                            }
                            break;

                    }
                    if(TREE_CODE(TYPE_NAME(t))==TYPE_DECL) {
                            /* ignore
                            printf(" Types %d \n",IDENTIFIER_POINTER(TYPE_NAME(t)));
                            **/
                    }
                    else if(TREE_CODE(TYPE_NAME(t))==IDENTIFIER_NODE){
                            strcpy(new_type_to_add,IDENTIFIER_POINTER(TYPE_NAME(t)));
                            add_simple_type(new_type_to_add,type_num,0,fn_name);
                            add_complex_type(t,l,type_num );
                    };
            }
    }
};

/**********************************************/
int process_array_pointer(tree t,tree l, char* fn_name){
/* t is tree_type */
tree st , ll;
char tm[TYPE_LENGTH], hld_str[TYPE_LENGTH];
    unsigned tn;

    tn=TYPE_UID(t);
    strcpy(tm,"");
```

```
        st=copy_node(t);
        TREE_CHAIN(st)=TREE_CHAIN(t);
        ll=copy_node(l);
        TREE_CHAIN(ll)=TREE_CHAIN(l);

        if(st==NULL)
                return;


        if(TYPE_EXISTS(tn))
                return;

        while(1) {

                if(st==NULL)
                        break;
                if(TREE_CODE(st)==POINTER_TYPE) {
                        strcat(tm,"ptr-to-");
                        st=TREE_TYPE(st);
                        continue;
                } else if (TREE_CODE(st)==ARRAY_TYPE){
                        strcat(tm,"array-of-");
                        st=TREE_TYPE(st);
                        continue;
                } else if(TREE_CODE(st)==FUNCTION_TYPE){
                        strcat(tm,"function-[rtns]-");
                        st=TREE_TYPE(st);
                        continue;
                } else if(IS_SIMPLE(st)) {
                        strcat(tm,get_code_name(TREE_CODE(st)));
                        break;
                }
                else {
                        strcat(tm,return_simple_type(st,ll));
                        break;
                }

        };
        add_simple_type(tm,tn,0,fn_name);

}
/*********************************************/
int add_complex_type(tree t, tree origin, unsigned type_num ){
tree f, f_type;
char temp_name[NAME_LENGTH];
char type_to_add[TYPE_LENGTH];
unsigned t_num, j, i;
static FIELD_LIST temp_field;

        for(i=0;i< MAX_VAR_NUM;i++) {
                if(s_data_type[i].data_type_id==type_num) {
                        j=i; break;
                };
        }

        for(f=TYPE_FIELDS(t);f; f=TREE_CHAIN(f)) {
                f_type=TREE_TYPE(f);
                t_num=TYPE_UID(f_type);

                if (IDENTIFIER_POINTER(DECL_NAME(f))!=NULL)
                        strcpy(temp_name,IDENTIFIER_POINTER(DECL_NAME(f)));
                else
```

```
                        strcpy(temp_name,"");

                /** add field first field_name & type-num **/
                temp_field=make_field(temp_name,t_num);
                if(add_field_to_list(s_data_type[j].fields, temp_field)==0)
                        s_data_type[j].fields=temp_field;
                /* add field type is not exsit **/
                if(TYPE_EXISTS(t_num))
                        continue;

                if(ARRAY_POINTER(f_type))
                        process_array_pointer(f_type, origin,"");
                else if(COMPLEX_T_N(f_type)) {
                        if(TYPE_NAME(f_type)==NULL)
                                strcpy(type_to_add,temp_name);
                        else {
                                if(IDENTIFIER_POINTER(TYPE_NAME(f_type))!=NULL)

        strcpy(type_to_add,IDENTIFIER_POINTER(TYPE_NAME(f_type)));
                                else strcpy(type_to_add,"");
                        };
                        add_simple_type(type_to_add,t_num,0,"");
                        add_complex_type(f_type,f,t_num);
                };
        };


}
/***********************************************************/
char * return_type_name(unsigned id ){
int i;
        for(i=0;i < MAX_VAR_NUM;i++){
                if (id== s_data_type[i].data_type_id ) {
                        return(s_data_type[i].data_type_name);
                };
        }
        return ("");
}
/*********************************************/
int initialize_data_types(){
int i;
        for(i=0;i < MAX_VAR_NUM;i++){

                /* simple data type */
                strcpy(s_data_type[i].data_type_name,"");
                s_data_type[i].def_module=0;
                strcpy(s_data_type[i].module_name,"");
                s_data_type[i].data_type_id=0;
                s_data_type[i].type_link_num=0;
                s_data_type[i].fields=NULL;

        };
};
/*****************************************************************/
int add_simple_type(char *new_type_name,unsigned type_id,
                int link_num , char * def_in_mod){

int free_data_slot, i ,add_flag ;
        add_flag=0;
        /* search if type id is already added **/

        for(i=0;i< MAX_VAR_NUM;i++){
```

```
                if(s_data_type[i].data_type_id== type_id)
                        add_flag=1;
        };
        if (add_flag==1) {
                return;
        };
        /*get first free slot **/

        free_data_slot=MAX_VAR_NUM;

        for(i=0;i< MAX_VAR_NUM;i++) {
                if(s_data_type[i].data_type_id==0) {
                        free_data_slot=i;
                        break;
                };
        };
        if(free_data_slot < MAX_VAR_NUM) {
                i= free_data_slot;
                strcpy(s_data_type[i].data_type_name,new_type_name);
                s_data_type[i].data_type_id= type_id;
                strcpy(s_data_type[i].module_name,def_in_mod);
                s_data_type[i].type_link_num= link_num;
        }
        else {
                printf("\nSystem can NOT accept any more TYPES\n");
                return -1;
        };

};
/*****************************************************************/
unsigned return_type_num(char *p_type){

int i;
 int rtn_num=0;
        if (strcmp(p_type,"")==0 )
                return rtn_num;
        for(i=0;i< MAX_VAR_NUM;i++){
                if(strcmp(s_data_type[i].data_type_name, p_type)==0){
                        rtn_num=s_data_type[i].data_type_id;
                        break;
                };
        };
        return rtn_num;
};
/*****************************************************************/
int print_simple_type(){
int i,j ;
FIELD_LIST tf;
        printf("\n*******SIMPLE DATA TYPES ******************\n");
        printf("\nData-Num\t\tType-Name\t\t\n");
        for(i=0; i< MAX_VAR_NUM;i++){

                if(s_data_type[i].data_type_id==0) break;

                printf("\n  %d  \t  %s %s \n", s_data_type[i].data_type_id,
                s_data_type[i].data_type_name,
                s_data_type[i].module_name);

                if (s_data_type[i].fields!=NULL) {
                        tf= s_data_type[i].fields ;
                        while(tf){
                        printf("\t\t\t\t\t=> Fields %s  Type=> %d \n",tf->field_name, tf->data_type_id);
```

```
                        tf=tf->next;
                };

            }
    }
    printf("\n******** END OF DATA TYPES ***************\n");
};
/************************************************************/
int print_complex_type(){
int i;
    printf("\n******* COMPLEX DATA TYPE ************\n");
    printf("\nData-Name\t\tType-Name\t\t\n");
    for(i=0;i< MAX_VAR_NUM;i++){
    };
};
/**********************************************************/
VAR_LIST make_var( char *p_name, unsigned  p_type_id){
static VAR_LIST temp;
    temp= (VAR_LIST ) malloc(sizeof(var));
    if(temp==NULL) {
      printf("\nCAN NOT MAKE MEMEORY !!!!!!\n");
    abort();
     return NULL;
    }
    strcpy(temp->name,p_name);
    temp->data_type_id=p_type_id;
    temp->read=0;
    temp->write=0;
    temp->control=0;
    temp->pass_parm=0;
    temp->next=NULL;
    return temp;
}
/*******************************************/

FIELD_LIST make_field( char *f_name, unsigned  f_type_id){
static FIELD_LIST temp;

    temp= (FIELD_LIST ) malloc(sizeof(FIELD_DATA));
    strcpy(temp->field_name,f_name);
    temp->data_type_id=f_type_id;
    temp->next=NULL;
    return temp;
}
/*******************************************/
int add_var_to_list(VAR_LIST dest  , VAR_LIST src){
  int c=0;
  if (dest!=NULL) {

    for(dest;dest->next!=NULL;dest=dest->next) ;
    dest->next=src;
    return 1;
  }
  else {
    dest=src;
    return 0;
    };
}
/*******************************************/

int add_field_to_list(FIELD_LIST dest  , FIELD_LIST src){
  if (dest!=NULL) {
```

```
        for(dest;dest->next!=NULL;dest=dest->next) ;
        dest->next=src;
        return 1;
  }
  else {
        dest=src;
        return 0;
        };
}
/*******************************************/
print_var_list(VAR_LIST l){
        while(l) {
                printf("\n\t\t\t %s   Type= %s [%d] WRC %d %d %d  ",l->name,return_type_name(l-
>data_type_id),l->data_type_id,
                                l->write, l->read , l->control);
                l=l->next;
                } ;
}
/**********************************************/
int is_complex(char *p_type){
int rts;
char * str, *un;
        rts=(strstr(p_type,STRUCT_STR)!=NULL ||
                strstr(p_type,UNION_STR)!=NULL);
        return rts;
};
/***********************************************/
char * brace_name(char *p, char *lb,char *rb){
static char tmp_char[NAME_LENGTH + 10];
        strcpy(tmp_char,lb);
        strcat(tmp_char,p);
        strcat(tmp_char,rb);
return tmp_char;
};
/********************************************************/

CALL_LIST make_fun_call( char *p_name , PARA_LIST used_para ){
static CALL_LIST temp;

        temp= (CALL_LIST ) malloc(sizeof(SINGLE_FUN_CALL));
        strcpy(temp->called_name,p_name);
        temp->passed_args=used_para;
        temp->next=NULL;

        return temp;
}
/************************************************/
int add_call_to_list(CALL_LIST dest , CALL_LIST src){
  if (dest!=NULL) {
        for(dest;dest->next!=NULL;dest=dest->next) ;
        dest->next=src;
        return 1;
  }
  else {
        dest=src;
        return 0;
        };
}
/********************************************************/
print_fun_calls(CALL_LIST p_calls){

        while(p_calls){
```

```
		printf("\n\t\t :%s \n", p_calls->called_name);
			print_parameter_list(p_calls->passed_args);

		p_calls=p_calls->next;
	};
};

/***********************************************************/

PARA_LIST make_parameter_node( char *p_type , char * p_name , char* p_exp_var ){
static PARA_LIST temp;

	temp= (PARA_LIST ) malloc(sizeof(PARAMETERS));

	strcpy(temp->parameter_type,p_type);
	strcpy(temp->parameter_name,p_name);
	strcpy(temp->exp_vars,p_exp_var);
	temp->next_para=NULL;

	return temp;
}
/****************************************************************/
int add_para_to_list(PARA_LIST dest  , PARA_LIST src){
  if (dest!=NULL) {
	for(dest;dest->next_para!=NULL;dest=dest->next_para) ;
	dest->next_para=src;
	return 1;
  }
  else {
	dest=src;
	return 0;
	};
}
/*******************************************************************/
void print_parameter_list(PARA_LIST  p_list){

PARA_LIST p;
	p=p_list;
	while (p) {
		printf("\t Parm-Type %s  [%s]  [%s]\n ", p->parameter_type , p->parameter_name,
			p->exp_vars);
		p=p->next_para;
	};
};
/*******************************************************************/
char * filter_comp(tree p, char * parent_name ){
static char rtn_str [NAME_LENGTH * 4  ];
static char t_str [NAME_LENGTH * 4];
tree t;
t=p;

get_component_name(t,&t_str[0],&rtn_str[0]);
	strcpy(parent_name,t_str);
	return (rtn_str);

}

/**********************************************************************/
int is_expression ( tree p) {
	if(

		TREE_CODE(p)== ABS_EXPR ||
```

```
            TREE_CODE(p)== ADDR_EXPR ||
            TREE_CODE(p)== BIT_ANDTC_EXPR ||
            TREE_CODE(p)== BIT_AND_EXPR ||
            TREE_CODE(p)== BIT_IOR_EXPR ||
            TREE_CODE(p)== BIT_XOR_EXPR ||
            TREE_CODE(p)== BIT_NOT_EXPR ||
            TREE_CODE(p)== CALL_EXPR ||
            TREE_CODE(p)== CEIL_DIV_EXPR ||
            TREE_CODE(p)== CEIL_MOD_EXPR ||
            TREE_CODE(p)== COND_EXPR ||
            TREE_CODE(p)== CONVERT_EXPR ||
            TREE_CODE(p)== EQ_EXPR ||
            TREE_CODE(p)== EXACT_DIV_EXPR ||
            TREE_CODE(p)== FFS_EXPR ||
            TREE_CODE(p)== FIX_CEIL_EXPR ||
            TREE_CODE(p)== FIX_FLOOR_EXPR ||
            TREE_CODE(p)== FIX_ROUND_EXPR ||
            TREE_CODE(p)== FIX_TRUNC_EXPR ||
            TREE_CODE(p)== FLOOR_DIV_EXPR ||
            TREE_CODE(p)== FLOOR_MOD_EXPR ||
            TREE_CODE(p)== GE_EXPR ||
            TREE_CODE(p)== GT_EXPR ||
            TREE_CODE(p)== INIT_EXPR ||
            TREE_CODE(p)== LROTATE_EXPR ||
            TREE_CODE(p)== LSHIFT_EXPR ||
            TREE_CODE(p)== LT_EXPR ||
            TREE_CODE(p)== MAX_EXPR ||
            TREE_CODE(p)== MINUS_EXPR ||
            TREE_CODE(p)== MIN_EXPR ||
            TREE_CODE(p)== MODIFY_EXPR ||
            TREE_CODE(p)== MULT_EXPR ||
            TREE_CODE(p)== NEGATE_EXPR ||
            TREE_CODE(p)== NON_LVALUE_EXPR ||
            TREE_CODE(p)== NOP_EXPR ||
            TREE_CODE(p)== PLUS_EXPR ||
            TREE_CODE(p)== POSTDECREMENT_EXPR ||
            TREE_CODE(p)== POSTINCREMENT_EXPR ||
            TREE_CODE(p)== PREDECREMENT_EXPR ||
            TREE_CODE(p)== PREINCREMENT_EXPR ||
            TREE_CODE(p)== RDIV_EXPR ||
            TREE_CODE(p)== RDIV_EXPR ||
            TREE_CODE(p)== ROUND_MOD_EXPR ||
            TREE_CODE(p)== ROUND_MOD_EXPR ||
            TREE_CODE(p)== RROTATE_EXPR ||
            TREE_CODE(p)== RSHIFT_EXPR ||
            TREE_CODE(p)== TRUNC_DIV_EXPR ||
            TREE_CODE(p)== TRUNC_MOD_EXPR ||
            TREE_CODE(p)== TRUTH_ANDIF_EXPR ||
            TREE_CODE(p)== TRUTH_AND_EXPR ||
            TREE_CODE(p)== TRUTH_NOT_EXPR ||
            TREE_CODE(p)== TRUTH_ORIF_EXPR ||
            TREE_CODE(p)== TRUTH_OR_EXPR ||
            TREE_CODE(p)== TRUTH_XOR_EXPR ||
            TREE_CODE(p)== FLOAT_EXPR ||
            TREE_CODE(p)== NE_EXPR ||
            TREE_CODE(p)== EXPON_EXPR ||
            TREE_CODE(p)== COMPOUND_EXPR ||
            TREE_CODE(p)== LE_EXPR
) {
            return 1;
}
else {
```

```
		return 0;
	};
}
/*********************************************************************/

int is_single_operand ( tree p) {
	if(

		TREE_CODE(p)== ABS_EXPR ||
		TREE_CODE(p)== FFS_EXPR ||
		TREE_CODE(p)== FIX_CEIL_EXPR ||
		TREE_CODE(p)== FIX_FLOOR_EXPR ||
		TREE_CODE(p)== FIX_ROUND_EXPR ||
		TREE_CODE(p)== FIX_TRUNC_EXPR ||
		TREE_CODE(p)== NEGATE_EXPR ||
		TREE_CODE(p)== NON_LVALUE_EXPR ||
		TREE_CODE(p)== NOP_EXPR ||
		TREE_CODE(p)== TRUTH_NOT_EXPR ||
		TREE_CODE(p)== BIT_NOT_EXPR ||
		TREE_CODE(p)== CARD_EXPR  ||
		TREE_CODE(p)== FLOAT_EXPR ||
		TREE_CODE(p)== CONVERT_EXPR ||
		TREE_CODE(p)== REFERENCE_EXPR ||
		TREE_CODE(p)== ADDR_EXPR ||
		TREE_CODE(p)== RETURN_EXPR ||
		TREE_CODE(p)== ENTRY_VALUE_EXPR ||
		TREE_CODE(p)== EXIT_EXPR ||
		TREE_CODE(p)== LOOP_EXPR ||
		TREE_CODE(p)== CLEANUP_POINT_EXPR ||
		TREE_CODE(p)== LABEL_EXPR

	) {
		return 1;
	} else {
		return 0;
	};
}

/*********************************************************************
this function scans a tree for variables in the tree the result is
saved in temp_para_list through function add_Str_to_exp_var ***/

char * get_exp_var_list(tree p, int add ){
static char *rtn_name;
VAR_LIST t;
unsigned tot_size=0;

temp_var_list=NULL;
/** get all vars in parameters including control **/

get_exp_var1(p,2,1);

t=temp_var_list;
/** calculate the size of the string **/
while(t) {
	tot_size=tot_size + sizeof(t->name) + 1;
	t=t->next;
};
/** concatenate the var-names in one string **/

if(tot_size!=0) {
	tot_size= tot_size+1;
```

```
		t=temp_var_list;
		rtn_name= (char*) malloc(tot_size);
		if(rtn_name==0)
			return ("");
		rtn_name[0]='\0';
		while(t){
			strcat(rtn_name,t->name);
			t=t->next;
			if(t!=NULL)
				strcat(rtn_name,",");
		};
}else {
		rtn_name= (char*) malloc(2);
		strcpy(rtn_name,"");
};
temp_var_list=NULL;

get_exp_var1(p,0,0); /* do not get function names */
t=temp_var_list;

while(t) {
		if(add==1)
			modify_var(t->name,global_num_of_fun,PASS_PARM,t->data_type_id);
		t=t->next;
};


return (rtn_name);
};
/**************************************************************************/
int add_str_to_exp_var(char *p_str){
		if(strstr(temp_para_exp_var,p_str)==NULL) {
			strcat(temp_para_exp_var,p_str);
			strcat(temp_para_exp_var,",");
		};
}
/*******************************************************************/
char * get_code_name(int i){
return (string_tree_code_type[i]);
};
/*******************************************************************/
int process_enum(tree t, tree origin){
tree f, ft;

		for(f=TYPE_FIELDS(t);f; f=TREE_CHAIN(f)) {
			ft=TREE_TYPE(f);
		}
};
/********************************************************************/

int var_exist( char *var_name, int fun_num, int scope) {
VAR_LIST temp;
int exist_flag=0;
		if (scope==0)   /* check local vars */
			temp=fun_vec[fun_num].local_var;
		if(scope==1)
			temp=fun_vec[fun_num].para_list;

		if(scope==2)
			temp=fun_vec[fun_num].global_var_used;

		exist_flag=0;
```

```
        while(temp){
                if(strcmp(temp->name,var_name)==0){
                        exist_flag=1;
                        break;
                }
                temp=temp->next;
        };
        return(exist_flag);

};

/*************** ***************************/
int modify_var(char * var_name, int fun_num, ACCESS_TYPE act, unsigned type_id){
VAR_LIST temp;
int flag;

        /* check for local & global variables used in this function
        if not exist then create new global-var **/

        flag=0;

        if(strcmp(var_name,"")==0)
                return;

        if(var_exist(var_name ,fun_num,0 )!=0 ) {
                temp=fun_vec[fun_num].local_var;
        }
        else if(var_exist(var_name ,fun_num,1 )!=0 ) {
                temp=fun_vec[fun_num].para_list;
        }
        else if(var_exist(var_name,fun_num,2)!=0 ) {
                temp=fun_vec[fun_num].global_var_used;
        }
        else {

                temp=make_var(var_name,type_id);
                if(add_var_to_list(fun_vec[fun_num].global_var_used,temp)==0)
                        fun_vec[fun_num].global_var_used=temp;

        }
        while(temp) {
                if(strcmp(temp->name,var_name)==0){
                        if(act==WRITE) {
                                temp->write= temp->write + 1;
                        }
                        if(act==READ) {
                                temp->read= temp->read + 1;
                        }
                        if(act==CONTROL) {
                                temp->control= temp->control + 1;
                        }
                        if(act==PASS_PARM) {
                                temp->pass_parm= temp->pass_parm + 1;
                        }

                }
                temp=temp->next;
        }
};
/**************************************************************************/
int process_cond_exp(tree cond ,int  fun_num){
```

```
char temp_name[NAME_LENGTH];

strcpy(temp_name,fun_vec[fun_num].name);
temp_var_list=NULL;
get_exp_var1(cond,1,0);
while(temp_var_list) {
                    modify_var(temp_var_list->name,fun_num,CONTROL,
                             temp_var_list->data_type_id);

                    temp_var_list=temp_var_list->next;

};
};
/****************************************************************************/

int process_read_write_exp(tree p_var, int  fun_num,ACCESS_TYPE act){
char temp_name[NAME_LENGTH];
char vmp_name[NAME_LENGTH];

if(TREE_CODE(p_var)==VAR_DECL)
          strcpy(vmp_name,IDENTIFIER_POINTER(DECL_NAME(p_var)));

else      strcpy(vmp_name,"");

strcpy(temp_name,fun_vec[fun_num].name);

temp_var_list=NULL;
get_exp_var1(p_var,0,0):
while(temp_var_list) {
                    modify_var(temp_var_list->name,fun_num.act,
                             temp_var_list->data_type_id);
                    temp_var_list=temp_var_list->next;
};
/** if WRITE request and LHS is array-ref=> make-array indexing as READ
since it will not be countered in write request. **/
temp_var_list=NULL;

if(act==WRITE && TREE_CODE(p_var)==ARRAY_REF) {
          get_exp_var1(TREE_OPERAND(p_var,1),0,0);

          while(temp_var_list) {
                    modify_var(temp_var_list->name,fun_num,READ,
                             temp_var_list->data_type_id);

                    temp_var_list=temp_var_list->next;
          };
};

};

/****************************************************************************/
/* cond_type=1 => only_condition_var 2=>all control variables 0=> op1&op2 only) **/
/* include_func_name=1 => includes names of functions as well **/
void get_exp_var1(tree p, int cond_type , int include_func_name) {
tree l,r,m, dr;
VAR_LIST t;
char comp_name[NAME_LENGTH];
char wfield_name[4* NAME_LENGTH];
last_cond_type=cond_type;

m=NULL; r=NULL; l=NULL;
```

```
if(is_expression(p)==1 || TREE_CODE(p)==ARRAY_REF) {

        if(is_single_operand(p)==1) {
                l=TREE_OPERAND(p,0);
                r=NULL; m=NULL;
        }
        else if(TREE_CODE(p)==COND_EXPR) {
                if(cond_type==1) {
                        l=TREE_OPERAND(p,0);
                        r=NULL; m=NULL;

                }
                else if(cond_type==2) {
                        l=TREE_OPERAND(p,0);
                        r=TREE_OPERAND(p,1);
                        m=TREE_OPERAND(p,2);
                } else { /* cond_type=0 **/
                        l=TREE_OPERAND(p,1);
                        r=TREE_OPERAND(p,2);
                        m=NULL;
                };

        }
        else if(TREE_CODE(p)==ARRAY_REF){
                l=TREE_OPERAND(p,0);
                r=NULL; m=NULL;
        }
        else {
                l=TREE_OPERAND(p,0);
                r=TREE_OPERAND(p,1);
                m=NULL;
        };

        if(l!=NULL )
                get_exp_var1(l, cond_type, include_func_name);

        if(r!=NULL)
                get_exp_var1(r, cond_type, include_func_name);

        if(m!=NULL)
                get_exp_var1(m, cond_type, include_func_name);
        return;

} else
        l=p;

if(l!=NULL) {
        extend_var_list(l, include_func_name);
}
if(r!=NULL){
        extend_var_list(r, include_func_name);
};

if(m!=NULL){
        extend_var_list(m, include_func_name);
        };

};
/****************************************************************/
int extend_var_list( tree p, int include_func) {
tree dr, l=p;
char comp_name[NAME_LENGTH];
```

```
char wfield_name[4* NAME_LENGTH];
VAR_LIST t;
unsigned type_num;

if(TREE_CODE(l)==VAR_DECL || TREE_CODE(l)==PARM_DECL ||
  TREE_CODE(l)==FIELD_DECL){

			type_num=TYPE_UID(TREE_TYPE(l));
			t=make_var(return_name(l),type_num);
			if(add_var_to_list(temp_var_list,t)==0)
				temp_var_list=t;
}
else if(TREE_CODE(l)==FUNCTION_DECL) {
		if(include_func==1) {
				type_num=TYPE_UID(TREE_TYPE(TREE_TYPE(l)));
			t=make_var(return_name(l),type_num);
			if(add_var_to_list(temp_var_list,t)==0)
				temp_var_list=t;
		}
}
else if(TREE_CODE(l)==COMPONENT_REF) {
			strcpy(comp_name,"");
			strcpy(wfield_name,"");
			get_component_name(l,&comp_name[0],&wfield_name[0]);
			t=make_var(comp_name,TYPE_UID(TREE_TYPE(l)));
			if(add_var_to_list(temp_var_list,t)==0)
				temp_var_list=t;
}  else if(TREE_CODE(l)==ARRAY_REF) {

			strcpy(comp_name,IDENTIFIER_POINTER(DECL_NAME(TREE_OPERAND(l,0))));
			t=make_var(comp_name,TYPE_UID(TREE_TYPE(l)));
			if(add_var_to_list(temp_var_list,t)==0)
				temp_var_list=t;

} else if(TREE_CODE(l)==INDIRECT_REF){
			strcpy(comp_name,"");
			strcpy(wfield_name,"");
			get_component_name(l,&comp_name[0],&wfield_name[0]);
			t=make_var(comp_name,TYPE_UID(TREE_TYPE(l)));
			if(add_var_to_list(temp_var_list,t)==0)
				temp_var_list=t;
}
} ;


/************************************************************/
void get_component_name(tree p ,char * parent_name, char* full_name ){
strcpy(full_field_name,"");
strcpy(str_name,"");
break_comp(p);
strcpy(parent_name,str_name);
strcpy(full_name,str_name);
strcat(full_name,full_field_name);

};
/************************************************************/

void break_comp(tree p) {
tree t, t0, t1;
char str_holder[NAME_LENGTH *4];
t=p;
```

```
        if (TREE_CODE(t)==COMPONENT_REF) {
                t1=TREE_OPERAND(t,1);
                t0=TREE_OPERAND(t,0);

                strcpy(str_holder,".");
                strcat(str_holder,return_name(t1));
                strcat(str_holder,full_field_name);
                strcpy(full_field_name,str_holder);


                 break_comp(t0);
        };

        if(TREE_CODE(t)==INDIRECT_REF) {
                t0=TREE_OPERAND(t,0);
                break_comp(t0);
        };
        if(TREE_CODE(t)==NOP_EXPR)  {
          t0=TREE_OPERAND(t,0);
                break_comp(t0);
        };
        if(TREE_CODE(t)==VAR_DECL ||
          TREE_CODE(t)==PARM_DECL ||
          TREE_CODE(t)==FUNCTION_DECL ){
                strcat(str_name,return_name(t));
        };
        if(TREE_CODE(t)==ARRAY_REF)
                break_comp(TREE_OPERAND(t,0));

        if(is_expression(t))
                get_exp_var1(t,last_cond_type,0); /* zero since it must be a var */
}
/*************************************************************************/
int mk_parm(tree t, int fun_num ){
tree v,p;
VAR_LIST t_v;
unsigned t_uid;
char temp_name[NAME_LENGTH];

if(TREE_CODE(t)==TREE_LIST){

        for(v=t;v;v=TREE_CHAIN(v)){
                p=TREE_VALUE(v);
        }
}
else {
        for(p=t;p;p=TREE_CHAIN(p)){
                strcpy(temp_name,return_name(p));
                t_uid=TYPE_UID(TREE_TYPE(p));
                t_v=make_var(temp_name,t_uid);
                if(add_var_to_list(fun_vec[fun_num].para_list,t_v)==0)
                        fun_vec[fun_num].para_list=t_v;

        }
}

};
/*********************************************************************/
int dump_data(char* mod_name, char *sys_name) {
FILE *fpfun, *fpcall, *fplcl, *fpug, *fptype;
FILE *fpfld, *fpparm, *fparg, *fpglbvar;
char        file_name[NAME_LENGTH];
```

```
char    fun_id[NAME_LENGTH *2];
char    call_id[NAME_LENGTH *2];
char temp_field1[NAME_LENGTH *2];
char temp_field2[NAME_LENGTH *2];
char temp_field3[NAME_LENGTH *2];
char temp_field4[NAME_LENGTH *2];
char temp_field5[NAME_LENGTH *2];
char temp_field6[NAME_LENGTH *2];
char temp_field7[NAME_LENGTH *2];
int last_call_id;
tree glob, temp_tree, t;
CALL_LIST fc;
FIELD_LIST f;
PARA_LIST p_list;
VAR_LIST var, parm, used_gl;
int i,j,p;
unsigned temp_id;

glob=getdecls();
last_call_id=get_last_call_id() + 1;
/********* output global variables **/
strcpy(file_name,"globvar.dat");
if((fpglbvar=fopen(file_name,"a"))==NULL) {
        if((fpglbvar=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
} else {

quote_str(sys_name,&temp_field1[0]);
quote_str(mod_name,&temp_field2[0]);
for(t=nreverse(glob);t;t=TREE_CHAIN(t)){

        strcpy(temp_field3,return_name(t));
        if(strcmp(temp_field3,"")==0)
                continue;
        quote_str(&temp_field3[0],&temp_field4[0]);
        if(TREE_CODE(t)==VAR_DECL) {
                i=(DECL_EXTERNAL(t)==0);
                fprintf(fpglbvar,"%s %s %s %d %d \n", temp_field1,temp_field2,
                        temp_field4, TYPE_UID(TREE_TYPE(t)),i);
                var=make_var(temp_field3,TYPE_UID(TREE_TYPE(t)));
                if(add_var_to_list(global_var_list,var)==0)
                        global_var_list=var;
        };

}
fclose(fpglbvar);
};

/********* output function files **/
strcpy(file_name,"fundef.dat");
if((fpfun=fopen(file_name,"a"))==NULL) {
        if((fpfun=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};

/********* output local_variable **/
strcpy(file_name,"lclvars.dat");
if((fplcl=fopen(file_name,"a"))==NULL) {
        if((fplcl=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};
```

```
/********* output format parameter files **/
strcpy(file_name,"formparm.dat");
if((fpparm=fopen(file_name,"a"))==NULL) {
        if((fpparm=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};

/********* output global vars used in function body **/
strcpy(file_name,"usedglob.dat");
if((fpug=fopen(file_name,"a"))==NULL) {
        if((fpug=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};

/********* output function calls **/
strcpy(file_name,"funcall.dat");
if((fpcall=fopen(file_name,"a"))==NULL) {
        if((fpcall=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};

/********* output passed args **/
strcpy(file_name,"argpassd.dat");
if((fparg=fopen(file_name,"a"))==NULL) {
        if((fparg=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};

/********* output data types **/
strcpy(file_name,"datatype.dat");
if((fptype=fopen(file_name,"a"))==NULL) {
        if((fptype=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};

/********* output data fields **/
strcpy(file_name,"fldtype.dat");
if((fpfld=fopen(file_name,"a"))==NULL) {
        if((fpfld=fopen(file_name,"w"))==NULL)
         printf("\nCAN NOT OPEN or CREATE %s File...\n",file_name);
};


/*******************************/
for(i=0;i< MAX_NUM_OF_FUN;i++) {
  if(strcmp(fun_vec[i].name,"")==0)
        break;
  quote_str(sys_name,&temp_field1[0]);
  quote_str(mod_name,&temp_field2[0]);


  strcpy(temp_field4,sys_name);
  strcat(temp_field4,mod_name);
  strcat(temp_field4,fun_vec[i].name);
  quote_str(temp_field4,&fun_id[0]);

  quote_str(fun_vec[i].name,&temp_field3[0]);

  fprintf(fpfun,"%s %s %s %1d %d %s \n",temp_field1,temp_field2,
        temp_field3,fun_vec[i].no_of_stmt,fun_vec[i].return_data_type_id, fun_id);

        /**output local variables **********************/
```

```
if(fun_vec[i].local_var!=NULL){
        var=fun_vec[i].local_var;
        while(var){
                quote_str(var->name,&temp_field4[0]);
                fprintf(fplcl,"%s %s %ld %ld %ld %ld %ld \n",fun_id,
                        temp_field4,var->data_type_id,var->read,var->write,var->control,
                        var->pass_parm);
                var=var->next;
        }
};
  /*****output formal parameters *****************/
if(fun_vec[i].para_list!=NULL){
        parm=fun_vec[i].para_list; j=1;
        while(parm){
                quote_str(parm->name,&temp_field4[0]);
                fprintf(fpparm,"%s %s %ld %ld %ld %ld %ld %ld \n",fun_id,
                        temp_field4,j,parm->data_type_id,parm->read,parm->write,parm->control,
                        parm->pass_parm);
                parm=parm->next;
                j=j+1;
        }
}

  /*****output GLOBAL VARIABLES USED IN FUNC BODY *****************/

if(fun_vec[i].global_var_used!=NULL){
        var=fun_vec[i].global_var_used;
        while(var){
                quote_str(var->name,&temp_field4[0]);
                temp_id=get_type_id(var->name);

                if(temp_id==0)
                        temp_id=var->data_type_id;
                fprintf(fpug,"%s %s %ld %ld %ld %ld %ld \n",fun_id,
                        temp_field4,temp_id,var->read,var->write,var->control, var->pass_parm);
                var=var->next;
        }
};

  /*****output fun calls  *****************/

if(fun_vec[i].fun_calls!=NULL){
        fc=fun_vec[i].fun_calls;
        j=1;
        while(fc){

                quote_str(fc->called_name,&temp_field4[0]);
                fprintf(fpcall,"%s %s %d %d \n",fun_id,temp_field4,j, last_call_id);

                if(fc->passed_args!=NULL){
                        p_list=fc->passed_args;
                        p=1;
                        while(p_list){
                                quote_str(p_list->parameter_type,&temp_field5[0]);
                                quote_str(p_list->parameter_name,&temp_field6[0]);
                                quote_str(p_list->exp_vars,&temp_field7[0]);

                                fprintf(fparg,"%s %s %ld %d %s %s %s %d \n",fun_id,temp_field4,j,
                                        p,temp_field5,temp_field6,temp_field7, last_call_id);
                                p_list=p_list->next_para;
                                p=p+1;
```

```
                    };
                };

                fc=fc->next;
                j=j+1;
                last_call_id= last_call_id + 1;
        }
  };

}; /* end of function scanning **/

/*** output data types and their fields if exist ***/

quote_str(sys_name,&temp_field1[0]);
quote_str(mod_name,&temp_field2[0]);

for(i=0; i< MAX_VAR_NUM;i++) {
        if(s_data_type[i].data_type_id==0)
                break;
        /* module_name below  represent function name type is def inside */
        quote_str(s_data_type[i].module_name,&temp_field3[0]);
        quote_str(s_data_type[i].data_type_name,&temp_field4[0]);
        fprintf(fptype,"%s %s %s %s %d \n",temp_field1,temp_field2,
                 temp_field3,temp_field4, s_data_type[i].data_type_id);

        if(s_data_type[i].fields!=NULL) {
                f=s_data_type[i].fields;
                quote_str(s_data_type[i].data_type_name,&temp_field4[0]);
                while(f){
                        quote_str(f->field_name,&temp_field5[0]);
                        fprintf(fpfld,"%s %s %s %s %d %s %d \n",temp_field1,temp_field2,
                                temp_field3,temp_field4, s_data_type[i].data_type_id,
                                temp_field5, f->data_type_id);
                        f=f->next;
                };

        };
}

write_last_call_id(last_call_id);
fclose(fpfun);
fclose(fptype);
fclose(fpfld);
fclose(fpparm);
fclose(fplcl);
fclose(fpug);
fclose(fpcall);
fclose(fparg);
};

/*********************************************************************/
int quote_str(char *src, char *dest){
int l,i;

dest[0]='"';
l=strlen(src) ;
for(i=1;i<= l ;i++)
        dest[i]=src[i-1];
dest[i]='"';
dest[i+1]='\0';
```

```
};

/***********************************************************************/
int get_system_name(char* s_name){
FILE *sfp;
int l;
sfp=fopen("sysfile","r");
if (sfp!=NULL) {
          fgets(s_name,NAME_LENGTH,sfp);
          l=strlen(s_name) - 1;
          s_name[l]='\0';
}
else strcpy(s_name,"NO_NAME");
fclose(sfp);
};

/***********************************************************************/
int get_last_call_id(){
FILE *sfp;
int l=0;
sfp=fopen("callid","r");
if (sfp!=NULL) {
          fscanf(sfp,"%d",&l);
          fclose(sfp);
}
return l;
};


/***********************************************************************/
int write_last_call_id(int p){
FILE *sfp;

sfp=fopen("callid","w");
if (sfp!=NULL) {
          fprintf(sfp,"%d",p);
          fclose(sfp);
}

};

/***********************************************************************/
char * single_fun_id(char *sys_name, char *fun_name){
static char rtn_fun_id[2 * NAME_LENGTH];
          strcpy(rtn_fun_id,"");
          strcat(rtn_fun_id,sys_name);
          strcat(rtn_fun_id,fun_name);

 return(rtn_fun_id);
};

/*****************************************************/
unsigned get_type_id(char *v_name){
char tmp_name[NAME_LENGTH];
VAR_LIST t_var;

t_var=global_var_list;
while (t_var) {

                    if(strcmp(t_var->name,v_name)==0) {
                              return t_var->data_type_id;
```

```
			}
		t_var=t_var->next;
	}

	return 0;
	}
```

Appendix C

This appendix contains three figures that are the listings of three programs. The first program is the routine that uncover all parameter to parameter aliases. The second program is the implementation of Dhama's method,

The third one is the implementation of cover-co-efficient metrics.

Figure 1. List of uncovering parameter aliases.

---

1. Uncover Aliases

The following is the listing of the routines used to uncover parameter-to-parameter aliases. The source code is written in Progress 4GL.

```
/** PROCEDURE THAT uncover parameter aliases**/

/**********************************************/
DEF TEMP-TABLE worklist
        FIELD   caller-id LIKE function.fun-id format 'x(35)'
        FIELD   caller-pos  AS INT
        FIELD   called-id LIKE function.fun-id format 'x(35)'
        FIELD   finished as logical
        FIELD   called-pos  AS INT.
DEF BUFFER WL FOR worklist.
DEF TEMP-TABLE fl
        FIELD   caller-id LIKE function.fun-id format 'x(20)'
        FIELD   caller-pos  AS INT
        FIELD   called-id LIKE function.fun-id format 'x(20)'
        FIELD   finished as logical
        FIELD   called-pos  AS INT.


DEF BUFFER PARM-BUF FOR fun-parm.


DEF VAR s-name AS CHAR.
DEF VAR m-name AS CHAR.
```

```
DEF VAR include-sys  AS LOGICAL.
DEF VAR is-sys  AS LOGICAL.
DEF VAR callee-fun-id LIKE function.fun-id.

/* below is user defined names of */
s-name='GCMS'.
m-name='tar'.
include-sys=no.


FOR EACH function WHERE sys-name= s-name and CAN-DO(m-name,module-name)
NO-LOCK by name  :

   FOR EACH p_alias WHERE p_alias.fun-id= function.fun-id:
      DELETE p_alias.
   END.

   FOR EACH fun-parm of function no-lock:
   if fun-parm.pass-parm=0 then
      next.

   FOR EACH fun-call WHERE fun-call.fun-id = function.fun-id by call-id:
      is-sys=CAN-FIND(FIRST sys-fun WHERE sys-fun.sys-name= function.sys-name
and
            sys-fun.name=fun-call.callee).
      IF is-sys=YES THEN NEXT.

      RUN get_fun_id( OUTPUT callee-fun-id, s-name,function.module-name, fun-
call.callee).
      FOR EACH passed-arg where passed-arg.call-id= fun-call.call-id no-lock by
position :
         IF passed-arg.arg-type <> 'PARAMETER' THEN
            NEXT.
         IF passed-arg.arg-nam<> fun-parm.name THEN
            NEXT.

         FIND FIRST worklist WHERE
         worklist.caller-pos= fun-parm.position and
         worklist.caller-id= function.fun-id AND
         worklist.called-id= callee-fun-id AND
         worklist.called-pos= passed-arg.position NO-ERROR.

         IF NOT AVAILABLE worklist THEN DO:
            CREATE worklist.
            worklist.caller-id=function.fun-id.
            worklist.caller-pos= fun-parm.position.
```

```
        worklist.called-id= callee-fun-id.
        worklist.called-pos= passed-arg.position.
      END.
    END.
  END.

  FOR EACH WORKLIST:

    run get_first_level(worklist.called-id, worklist.called-pos).

      FOR EACH fl:
       FIND FIRST wl WHERE wl.caller-id= worklist.caller-id AND
             wl.caller-pos= worklist.caller-pos AND
             wl.called-id= fl.called-id and
             wl.called-pos= fl.called-pos no-error.
        if not available wl THEN DO:
        CREATE wl.
          wl.caller-id= worklist.caller-id.
          wl.caller-pos= worklist.caller-pos.
          wl.called-id= fl.called-id.
          wl.called-pos= fl.called-pos.
        end.
      END.
      worklist.finished=yes.

    END.

END.

    FOR EACH worklist :
      CREATE p_alias.
      assign
      p_alias.fun-id= worklist.caller-id
      p_alias.position= worklist.caller-pos
      p_alias.called-id= worklist.called-id
      p_alias.called-pos= worklist.called-pos.
      DELETE worklist.
    END.

END.


/*************************************/
PROCEDURE get_first_level:
DEF INPUT PARAMETER p-fun LIKE function.fun-id.
```

```
DEF INPUT PARAMETER p-pos AS INT.
DEF BUFFER fun-buf FOR function.
DEF BUFFER pram-buf for fun-parm.
DEF BUFFER call-buf FOR fun-call.
DEF BUFFER p-arg FOR passed-arg.
DEF VAR callee-id LIKE function.fun-id.
FOR EACH fl:
  DELETE fl.
END.

FIND fun-buf WHERE fun-buf.fun-id= p-fun.
FIND FIRST parm-buf of fun-buf  where parm-buf.position= p-pos.
if parm-buf.pass-parm=0 then
  return.

FOR EACH call-buf of fun-buf BY call-id:

    is-sys=CAN-FIND(FIRST sys-fun WHERE sys-fun.sys-name= fun-buf.sys-name
and
        sys-fun.name=call-buf.callee).
    IF is-sys=YES THEN NEXT.

    RUN get_fun_id( OUTPUT callee-id, fun-buf.sys-name,fun-buf.module-name, call-
buf.callee).


  FOr EACH p-arg WHERE p-arg.call-id= call-buf.call-id:
    IF p-arg.arg-type<>'parameter' then
      next.
    IF p-arg.arg-name <> parm-buf.name THEN
      NEXT.
    FIND FIRST fl WHERE
      fl.caller-id= fun-buf.fun-id AND
      fl.caller-pos= parm-buf.position and
      fl.called-id= callee-id AND
      fl.called-pos= p-arg.position NO-ERROR.

      IF NOT AVAILABLE fl THEN DO:
        CREATE fl.
        fl.caller-id=fun-buf.fun-id.
        fl.caller-pos= parm-buf.position.
        fl.called-id= callee-id.
        fl.called-pos= p-arg.position.
      END.
  END.
END.
```

```
END.

/********************************/
PROCEDURE get_fun_id:
DEF OUTPUT PARAMETER r-id LIKE function.fun-id.
DEF INPUT PARAMETER p-s AS CHAR.
DEF INPUT PARAMETER p-m AS CHAR.
DEF INPUT PARAMETER p-f AS CHAR.

DEF BUFFER t-fun FOR function.

/** check same module **/

FIND FIRST t-fun WHERE t-fun.sys-name= p-s AND t-fun.module-name=p-m AND
   t-fun.name = p-f NO-LOCK NO-ERROR.
IF NOT AVAILABLE t-fun THEN
   FIND FIRST t-fun WHERE t-fun.sys-name= p-s AND t-fun.name = p-f NO-LOCK
NO-ERROR.

IF NOT AVAILABLE t-fun THEN
   r-id= p-s + p-m + p-f.
ELSE
   r-id= t-fun.fun-id.


END PROCEDURE.
```

Figure 2. Listing that clculate Dhama's coupling metrics.

2. Dhama's Method

The following is the listing of the routines used to calculate the system coupling using Dhama's method. The source code is written in Progress 4GL.

```
/** PROCEDURE THAT CALCULATE DHAMA METHOD **/
DEF TEMP-table dhama
        field proc-name             like function.name
        field input-para-data       as int
        field input-para-control    as int
        field output-para-data      as int
        field output-para-control   as int
        field num-of-global-data    as int
        field num-of-global-control as int
        field num-of-called-mod     as int
        field num-of-calling-mod    as int
        field name-of-calling-proc  as char format 'x(80)'
        field mod-coupling          as DEC FORMAT ">>99.99<<".

DEF QUERY d-qry FOR dhama.
DEF BROWSE d-brw QUERY d-qry  DISP  dhama.proc-name dhama.mod-coupling
WITH 15 DOWN.
DEF VAR match-str AS CHAR.
DEF VAR i AS INT.
DEF VAR s-name AS CHAR.
DEF VAR m-name AS CHAR.
DEF VAR sys-cop AS DEC.
DEF VAR tot-cop AS DEC.
DEF BUFFER t-fun FOR function.
DEF VAR include-sys AS LOGICAL.
FOR each dhama:
  DELETE dhama.
END.


s-name='GCMS'.
m-name='tar'.
include-sys=no.

FOR EACH function WHERE sys-name= s-name and CAN-DO(m-name,module-name)
NO-LOCK :

   CREATE dhama.
```

```
dhama.proc-name=function.name.

FOR EACH fun-call WHERE fun-call.fun-id=function.fun-id no-lock:
  IF include-sys=NO AND CAN-FIND(FIRST sys-fun WHERE sys-fun.sys-name= s-
name AND sys-fun.name=fun-call.callee ) THEN
     NEXT.
  dhama.num-of-called-mod = num-of-called-mod + 1.

END.

FOR EACH fun-call WHERE fun-call.fun-id begins s-name and
  fun-call.callee = function.name no-lock:

    find t-fun where t-fun.fun-id= fun-call.fun-id no-lock.
    if NOT CAN-DO(m-name,t-fun.module-name) THEN NEXT.
    dhama.num-of-calling-mod = num-of-calling-mod + 1.
    dhama.name-of-calling-proc = name-of-calling-proc + "," + t-fun.name.
END.

FOR EACH fun-parm of function no-lock:
  if fun-parm.control-op>0 THEN DO:
     if(fun-parm.read-op >0) then
        dhama.input-para-control  =  dhama.input-para-control + 1.
     if fun-parm.write-op >0  then
        dhama.output-para-control  =  dhama.output-para-control + 1.
     if fun-parm.pass-parm >0  then
        dhama.output-para-control  =  dhama.output-para-control + 1.
  end.
  else do:
     if(fun-parm.read-op >0) then
        dhama.input-para-data  =  dhama.input-para-data + 1.
     if fun-parm.write-op >0  then
        dhama.output-para-data  =  dhama.output-para-data + 1.
     if fun-parm.pass-parm >0  then
        dhama.output-para-data  =  dhama.output-para-data + 1.

  end.

END.
FOR EACH usedglb of function no-lock:
  if usedglb.control-op>0 THEN
     dhama.num-of-global-control  =  dhama.num-of-global-control + 1.
  else
     if(usedglb.read-op >=1 or usedglb.write-op >=1 or usedglb.pass-parm>0) then
        dhama.num-of-global-data  =  dhama.num-of-global-data + 1.
```

```
    END.

    dhama.mod-coupling = dhama.input-para-data + (2 * dhama.input-para-control) +
        dhama.output-para-data + ( 2 * dhama.output-para-control) +
        dhama.num-of-global-data + (2 * dhama.num-of-global-control) +
        dhama.num-of-called-mod + dhama.num-of-calling-mod.
     IF(DHAMA.MOD-COUPLING>0) THEN
      DHAMA.MOD-COUPLING = 1 / DHAMA.MOD-COUPLING.
END.

output to "outdham".
put m-name SKIP(1).
i=0. tot-cop=0.
FOR EACH dhama:
tot-cop= tot-cop + dhama.mod-coupling.
i=i + 1.
put dhama.proc-name dhama.mod-coupling SKIP.
end.
sys-cop= tot-cop / i.
PUT SKIP(2)
"*** System Coupling **** " sys-cop.
output close.
```

Figure 3. List of program that calculates CC coupling metrics.

---

3. Cover-Coefficient Matrix

The following is the listing of the routines used to build and calculate the definition and coupling matrices of cover-coefficient method.
The source code is written in Progress 4GL.

```
PROCEDURE get_decimal:
DEF OUTPUT PARAMETER rtn-dec AS INT.
DEF INPUT PARAMETER G AS INT.
DEF INPUT PARAMETER C AS INT.
DEF INPUT PARAMETER S AS INT.
   rtn-dec =((G * 4) + (C * 2) + (S * 1)) + 1.
END PROCEDURE.
/**********************************************/
DEF TEMP-table I-row
          field proc-name  like function.name
          field fun-id     like function.fun-id
          field num        AS int
          field alpha      as dec format '>>9.999<<'.

DEF TEMP-table J-col
          field id         as char format 'x(20)'
          field type       as char
          field num        AS int
          field fun-id     like function.fun-id
          field beta       AS dec format '>>9.999<<'
          field is_complex   as logical init no
          INDEX num num
          index id id.

DEF TEMP-table I-j-link
          field i-num      AS int
          field j-num      as int
          field G-v        as int
          field C-v        as int
          field S-v        as int
          field link-v     as dec
          field row-value    as dec
          index i-j-num i-num j-num.

DEF TEMP-TABLE I-mat
          field num        AS int
          field name       as char
```

```
            index num num.

DEF TEMP-TABLE J-mat
            field num         AS int
            field name        as char
            index num num.

DEF TEMP-TABLE I-J-mat
            field i-num       AS int
            field j-num       AS int
            field I-J-value       AS DEC FORMAT '>>9.99<<'
            index num i-num j-num.


DEF BUFFER t-fun FOR function.
DEF BUFFER PARM-BUF FOR fun-parm.
DEF BUFFER j-buf FOR j-col.
DEF BUFFER i-buf for I-ROW.
DEF VAR temp-alpha as dec format '>>9.999<<'.

DEF VAR read-sum as int.
DEF VAR cntl-sum as int.
DEF VAR writ-sum as int.
DEF VAR i AS INT.
DEF VAR j AS INT.
DEF VAR G AS INT.
DEF VAR C AS INT.
DEF VAR S AS INT.
DEF VAR K AS INT.
DEF VAR N AS INT.
DEF VAR num-of-fun AS INT.
DEF VAR ik_value AS DEC.
DEF VAR jk_value AS DEC.
DEF VAR callee-G AS INT.
DEF VAR callee-C AS INT.
DEF VAR callee-S AS INT.
DEF VAR lcl-list AS CHAR.
DEF VAR glb-list AS CHAR.
DEF VAR prm-list AS CHAR.

DEF VAR temp-arg-type AS CHAR.
DEF VAR rtn-j-num LIKE j-col.num.
DEF VAR callee-rtn-j-num LIKE j-col.num.
DEF VAR temp-var-name AS CHAR.
DEF VAR temp-int AS INT .
DEF VAR temp-str AS CHAR.
```

```
DEF VAR s-name AS CHAR.
DEF VAR m-name AS CHAR.
DEF VAR temp-dec AS DEC.
DEF VAR temp-flag AS LOGICAL.
DEF VAR include-sys  AS LOGICAL.
DEF VAR has-arg AS LOGICAL.
DEF VAR is-sys  AS LOGICAL.
DEF VAR callee-fun-id LIKE function.fun-id.
DEF VAR tot-cp AS DEC FORMAT '>>9.999<<' LABEL "System Coupling".
DEF VAR avg-cp AS DEC FORMAT '>>9.999<<' LABEL "AVG  Coupling".

FOR EACH I-row:
  DELETE I-row.
END.

s-name='gcms'.
m-name='tar'.
include-sys=NO.
i=1.

FOR EACH function WHERE sys-name= s-name and CAN-DO(m-name,module-name)
NO-LOCK  BY function.name:

    run make-row(function.fun-id, function.name).

    FOR EACH usedglb of function:

       /* if only used as passed-parm then it will be included then at fun-caal sec*/
       if(usedglb.write-op=0 AND usedglb.read-op=0 AND usedglb.control-op=0) THEN
         NEXT.

       temp-flag=NO.

       FOR EACH j-col WHERE j-col.id= usedglb.name AND j-col.type='GV':
         temp-flag=YES.
         LEAVE.
       END.
       IF temp-flag=NO THEN
         run ADD-VAR-TO-J-COL(RECID(usedglb), function.fun-id,"GV").

    END.

END.


/** assign value of global-vars used directly by the system **/
```

```
FOR EACH function WHERE sys-name= s-name and CAN-DO(m-name,module-name)
NO-LOCK :

   FIND FIRST  i-row where I-row.fun-id=function.fun-id.


   FOR EACH usedglb of function:
      if(usedglb.write-op=0 AND usedglb.read-op=0 AND usedglb.control-op=0) THEN
        NEXT.
      FIND FIRST j-col WHERE  j-col.id =usedglb.name AND j-col.type='GV' no-
error.
      if not available J-COL then
        next.

     S=(if j-col.is_complex=YES THEN 1 ELSE 0).
     cntl-sum=0.
     read-sum=0.
     writ-sum=0.
     if usedglb.control-op>0 THEN DO:
      run get_decimal(OUTPUT temp-int, 1,1,S) .
      cntl-sum = temp-int * usedglb.control-op.

     END.
     if usedglb.write-op >0 THEN DO:
        run get_decimal(OUTPUT temp-int, 1,0,S) .
        writ-sum = temp-int * usedglb.write-op.
     END.
     if usedglb.read-op >0 THEN DO:
        run get_decimal(OUTPUT temp-int, 1,0,S) .
        read-sum =temp-int * usedglb.read-op.
     end.
     temp-int= read-sum + writ-sum + cntl-sum.
     run update_link( i-row.num, j-col.num, temp-int).
   END.

END.

/*
MESSAGE "Finish ADDING GLOBAL-VARIABLES " view-as alert-box.
 */

/*** list element participating in calling-scheme **/
```

```
FOR EACH function WHERE sys-name= s-name and CAN-DO(m-name,module-name)
NO-LOCK by name  :

   run get_fun_list(buffer function, OUTPUT lcl-list,OUTPUT glb-list,OUTPUT prm-
list).

   FOR EACH fun-call WHERE fun-call.fun-id = function.fun-id :

      is-sys=CAN-FIND(FIRST sys-fun WHERE sys-fun.sys-name= function.sys-name
and
            sys-fun.name=fun-call.callee).

      IF is-sys=YES  THEN DO:
         IF include-sys = NO THEN
            NEXT.
         ELSE DO: /* if sys-fun does not exist as i-row then create one */
               run make_sys_fun(function.sys-name,fun-call.callee).
               callee-fun-id= RETURN-VALUE.
         END.

      END.
      ELSE DO:
         RUN get_fun_id( OUTPUT callee-fun-id, s-name,function.module-name, fun-
call.callee).

         IF NOT CAN-FIND(FIRST i-buf WHERE i-buf.fun-id= callee-fun-id) THEN
              run make-row( callee-fun-id, fun-call.callee).

      END.

      has-arg=NO.

      FOR EACH passed-arg where passed-arg.call-id= fun-call.call-id no-lock by
position :
          has-arg=YES.
         case passed-arg.arg-type :


            WHEN 'VARIABLE' OR WHEN "FIELD" THEN DO:

                  temp-var-name=passed-arg.arg-name .
                  RUN IS_VAR_EXIST(output rtn-j-num,temp-var-name,passed-arg.arg-
type, function.fun-id).

                  FIND FIRST j-col WHERE j-col.num= rtn-j-num NO-ERROR.
                  IF NOT AVAILABLE j-col THEN
```

```
          NEXT.
        S=(IF j-col.is_complex THEN 1 ELSE 0).

        IF CAN-DO(lcl-list, temp-var-name) THEN DO:
          G=0.
          FIND local-var of function WHERE local-var.name=temp-var-name
no-lock.
          if local-var.control-op>0 then
            run get_decimal(OUTPUT temp-int, 0,1,S) .
          ELSE
            run get_decimal(OUTPUT temp-int, 0,0,S) .
        END.
        ELSE IF CAN-DO(glb-list, temp-var-name) THEN DO:

          FIND usedglb of function WHERE usedglb.name=temp-var-name no-
lock.
          IF usedglb.pass-parm=0 THEN
            NEXT. /** this is included in usedglb loop for write/read **/
          IF usedglb.control-op>0 THEN
            run get_decimal(OUTPUT temp-int, 1,1,S) .
          ELSE
            run get_decimal(OUTPUT temp-int, 1,0,S).
        END.
        ELSE NEXT.

        /** update -link of caller**/
        FIND FIRST i-row WHERE i-row.fun-id= function.fun-id.
        RUN update_link (i-row.num, j-col.num, temp-int).

        /* update link callee and its aliases */
        FIND FIRST i-buf WHERE i-buf.fun-id= callee-fun-id NO-ERROR.
        IF NOT AVAILABLE i-buf THEN DO:
            MESSAGE "i-buf" callee-fun-id view-as alert-box.
            NEXT.
        END.

        RUN update-called-FUN (function.fun-id,j-col.num,callee-fun-id, passed-
arg.position, S ).

    END.

    WHEN "CONSTANT" OR WHEN "EXPRESSION" THEN DO: /* find para-
of-callee and return its j-row-nu
                    if not avail then create one and return its j-num */

        IF is-sys=NO THEN DO:
```

```
FOR EACH parm-buf WHERE parm-buf.fun-id= callee-fun-id :
   IF parm-buf.position= passed-arg.position THEN DO:
      callee-C=(IF parm-buf.control-op>0 THEN 1 ELSE 0).
      run IS_VAR_EXIST(output rtn-j-num,parm-
buf.name,'PARAMETER', callee-fun-id).
      LEAVE.
   END.
END.
END.
ELSE DO:
   if include-sys=NO THEN
      NEXT.

FOR EACH  sf-parm WHERE sf-parm.fun-id= callee-fun-id by sf-
parm.position:

   IF sf-parm.position = passed-arg.position THEN DO:
     FIND FIRST j-col WHERE j-col.fun-id= callee-fun-id AND
        j-col.id = sf-parm.name NO-ERROR.
        IF NOT AVAILABLE j-col THEN DO: /* create j-col.*/
           temp-int=0.
           FOR EACH j-buf BY j-buf.num:
              if temp-int < j-buf.num THEN
                 temp-int= j-buf.num.
           END.
           temp-int = temp-int + 1.
           CREATE j-buf.
           ASSIGN
              j-buf.id= sf-parm.name
              j-buf.fun-id= callee-fun-id
              j-buf.type= 'RG'
              j-buf.is_complex=NO.
              j-buf.num = temp-int.
              rtn-j-num= temp-int.
        END.
        ELSE rtn-j-num= j-col.num.
        LEAVE.

   END.
END.

END.


/** update -link of caller **/
FIND FIRST j-col WHERE j-col.num= rtn-j-num NO-ERROR.
```

```
        IF NOT AVAILABLE j-col THEN DO:
           bell.
           message 'j-col' rtn-j-num function.name callee 'not-found'
           'is-sys' is-sys view-as alert-box error.
        END.

        FIND FIRST i-row WHERE i-row.fun-id= function.fun-id.

        /** caller is passing constant **/
        run get_decimal(OUTPUT temp-int, 0,0,0) .

        RUN update_link (i-row.num, j-col.num, temp-int).
        /* check callee now **/

           RUN update-called-FUN (function.fun-id,j-col.num,callee-fun-id, passed-
           arg.position, S  ).

      END. /* WHEN CONSTATN */

      WHEN "FUNCTION_CALL" THEN
        NEXT.

    END CASE.

  END. /* of passed-arg */

  if has-arg=NO THEN DO:

     G=1.    S=0.     C=0.
    callee-G=1. callee-S=0. callee-C=0.

    RUN IS_FUN_EXIST( OUTPUT rtn-j-num, callee-fun-id, fun-call.callee ).

    FIND FIRST j-col WHERE j-col.num = rtn-j-num .
    FIND FIRST i-row WHERE i-row.fun-id= fun-call.fun-id.
    FIND FIRST i-buf WHERE i-buf.fun-id= callee-fun-id.

    S=(IF j-col.is_complex=YES THEN 1 ELSE 0).
    callee-S= S.
    run get_decimal(OUTPUT temp-int, G,C,S) .
    RUN update_link (i-row.num, j-col.num, temp-int).
    RUN update_link (i-buf.num, j-col.num, temp-int).

  END.
END.
```

```
END.


FOR EACH i-row:
   num-of-fun = num-of-fun + 1.
   temp-alpha=0.

   FOR EACH i-j-link WHERE i-j-link.i-num= i-row.num.
      temp-alpha= temp-alpha + i-j-link.link-v.
   END.
   i-row.alpha= (if temp-alpha>0 THEN (1 / temp-alpha) ELSE 0).
END.

FOR EACH j-col:
   temp-alpha=0.
   FOR EACH i-j-link WHERE i-j-link.j-num= j-col.num.
      temp-alpha= temp-alpha + i-j-link.link-v.
   END.
   j-col.beta= (if temp-alpha>0 THEN (1 / temp-alpha) ELSE 0).
END.


DISP "Start Calculating CC Matrix" WITH FRAME CC_M view-as dialog-box.
pause 0.

FOR EACH I-MAT: DELETE I-MAT. END.
FOR EACH J-MAT: DELETE J-MAT. END.
FOR EACH I-J-MAT: DELETE I-J-MAT. END.

FOR EACH i-row by i-row.num:
   CREATE I-MAT. CREATE J-MAT.
   I-MAT.num = i-row.num. J-MAT.num= i-row.num.
   I-MAT.name= i-row.proc-name. J-MAT.name= i-row.proc-name.
END.
i=0.
FOR EACH j-col BY j-col.NUM:
   IF j-col.num> i THEN i=j-col.num.
END.
FOR EACH i-mat By num:
   FOR EACH j-mat BY num:
      CREATE i-j-mat.
      i-j-mat.i-num= i-mat.num.
      i-j-mat.j-num= j-mat.num.
   END.
END.
```

```
DEF BUFFER  d_ik  FOR i-j-link.
DEF BUFFER  d_jk  FOR i-j-link.

tot-cp=0. avg-cp=0.

FOR EACH i-mat By num:
   FIND FIRST i-row WHERE i-row.num= i-mat.num.
   temp-dec=0.

   FOR EACH i-j-mat WHERE i-j-mat.i-num= i-mat.num BY j-num:
      temp-dec=0.

      DO K=1 TO i:
         FIND j-col WHERE j-col.num= K.
         FIND FIRST d_ik WHERE d_ik.i-num=i-row.num     AND d_ik.j-num=K NO-
ERROR.
         ik_value=(IF AVAILABLE d_ik THEN d_ik.link-v ELSE 0).
         FIND FIRST d_jk WHERE d_jk.i-num=i-j-mat.j-num AND d_jk.j-num=K NO-
ERROR.
         jk_value=(IF AVAILABLE d_jk THEN d_jk.link-v ELSE 0).
         temp-dec= temp-dec + (ik_value * j-col.beta  * jk_value).
      END.

      i-j-mat.i-j-value = i-row.alpha * temp-dec.
   END.
END.


run send-to-excel.

/************************************************/
PROCEDURE update-called-fun:
DEF INPUT PARAMETER p-exclude-fun LIKE function.fun-id.
DEF INPUT PARAMETER j-pos      AS      INT.
DEF INPUT PARAMETER p-fun-id    LIKE    function.fun-id.
DEF INPUT PARAMETER p-pos      AS      INT.
DEF INPUT PARAMETER p-complex   AS      INT.

DEF BUFFER f-buf FOR function.
DEF BUFFER f2-buf FOR function.
DEF BUFFER parm-buf FOR fun-parm.
DEF BUFFER parm2-buf FOR fun-parm.
DEF VAR v-int as int.
DEF VAR cntl AS INT.
```

```
FIND f-buf WHERE f-buf.fun-id= p-fun-id .

FIND FIRST parm-buf WHERE parm-buf.fun-id= f-buf.fun-id AND
parm-buf.position= p-pos.

cntl=(if parm-buf.control-op>0 THEN 1 ELSE 0).

FIND FIRST i-row WHERE i-row.fun-id= p-fun-id NO-ERROR.
IF NOT AVAILABLE i-row THEN
   run make-row(p-fun-id, f-buf.name).

FIND FIRST i-row WHERE i-row.fun-id= p-fun-id NO-ERROR.

run get_decimal(OUTPUT v-int,0,cntl,p-complex).

run update_link(i-row.num, j-pos, v-int).

FOR EACH p_alias WHERE p_alias.fun-id= f-buf.fun-id AND p_alias.position= p-pos
no-lock:
   if( p_alias.called-id= p-exclude-fun OR p_alias.called-id= p-fun-id) THEN
      NEXT.
   FIND FIRST f2-buf WHERE f2-buf.fun-id= p_alias.called-id no-error.
   if not available f2-buf then
      next.
   FIND FIRST i-row WHERE i-row.fun-id= f2-buf.fun-id no-error.
   if not available i-row then do:
        run make-row(f2-buf.fun-id, f2-buf.name).
        FIND FIRST i-row WHERE i-row.fun-id= f2-buf.fun-id .
   end.
   FIND FIRST parm2-buf WHERE parm2-buf.fun-id= f2-buf.fun-id AND
      parm2-buf.position= p_alias.called-pos.
   cntl=(if parm2-buf.control-op>0 THEN 1 ELSE 0).

   run get_decimal(OUTPUT v-int,0,cntl,p-complex).
   run update_link(i-row.num, j-pos, v-int).

END.


END PROCEDURE.

/***********************************************/
PROCEDURE make-row:
DEF INPUT PARAMETER passed-fun-id AS CHAR.
DEF INPUT PARAMETER passed-fun-name AS CHAR.
```

```
DEF VAR i AS INT.

i=0.
FOR EACH i-buf BY i-buf.num:
   i=i + 1.
END.
i= i + 1.
CREATE i-buf.
   i-buf.fun-id= passed-fun-id.
   i-buf.proc-name= passed-fun-name.
   i-buf.num= i.

END PROCEDURE.
/**********************************************/
procedure update_link:
def input parameter passed-i as int.
def input parameter passed-j as int.
def input parameter passed-value as dec.

   FIND FIRST  i-j-link WHERE i-j-link.i-num= passed-i AND
      i-j-link.j-num= passed-j NO-ERROR.
   IF NOT AVAILABLE i-j-link THEN DO:
      CREATE i-j-link.
      i-j-link.i-num= passed-i.
      i-j-link.j-num= passed-j.
   END.
/*MESSAGE "I=" passed-i SKIP
      "J=" passed-j SKIP
      "OLD-VALUEI=" i-j-link.link-v SKIP
      "NEW VALUE" passed-value
      VIEW-aS ALERT-BOX.*/


   i-j-link.link-v = i-j-link.link-v + passed-value.



end procedure.
/**********************************************/
PROCEDURE add-glob-in-j-row:
def PARAMETER buffer ug FOR usedglb.
DEF INPUT PARAMETER pass-fun-id LIKE function.fun-id.
DEF INPUT PARAMETER col-type AS CHAR.

DEF BUFFER j-buf for j-col.
```

```
DEF BUFFER buf-fun for function.
DEF VAR j as int.
DEF VAR is_str AS LOGICAL.



FIND buf-fun WHERE buf-fun.fun-id= pass-fun-id.

FIND FIRST j-col where j-col.id= ug.name AND j-col.type=col-type  NO-ERROR.

IF NOT AVAILABLE j-col THEN DO:

  j=1.
  FOR EACH j-buf:
   j=j + 1.
  end.

  CREATE j-col.
  j-col.id= ug.name.
  j-col.type=col-type.
  j-col.num = j.
  is_str=NO.
  run is_struct(OUTPUT  is_str,ug.type-id,buf-fun.sys-name,buf-fun.module-name,
  buf-fun.module-name).

  j-col.is_complex= is_str.

END.

END.

/************************************************/

PROCEDURE IS_STRUCT :
DEF OUTPUT PARAMETER rtn_v AS logical init no NO-UNDO.
DEF INPUT PARAMETER passed-type AS INT.
DEF INPUT PARAMETER passed-sys AS char.
DEF INPUT PARAMETER passed-mod AS char.
DEF INPUT PARAMETER passed-fun AS char.

DEF VAR temp-fun-name AS CHAR.
DEF VAR temp-type-name AS CHAR.
DEF VAR rtn-flag AS LOGICAL.
rtn_v= NO .

temp-fun-name= passed-fun.
```

```
FIND FIRST types WHERE types.type-id= passed-type AND
            types.sys-name=passed-sys AND
types.module-name=passed-mod  AND types.fun-name=temp-fun-name NO-LOCK NO-
ERROR.
IF NOT AVAILABLE types THEN DO:
        temp-fun-name="".
       FIND FIRST types WHERE types.type-id= passed-type AND types.sys-name =
       passed-sys AND   types.module-name=passed-mod  AND types.fun-name= temp-
       fun-name NO-LOCK NO-ERROR.

END.

IF NOT AVAILABLE types THEN DO:

   return.
END.

temp-type-name= types.type-name.


IF CAN-FIND(FIRST type-field WHERE type-field.parent-type-id= passed-type AND
type-field.sys-name=passed-sys AND
   type-field.module-name= passed-mod and type-field.fun-name= temp-fun-name)
THEN DO:

   rtn_v=YES.
   return.
END.

/** check for pointer and array type-name before making return 0 ****/

if(INDEX(types.type-name,'array-of-',1) = 0 AND
   INDEX(types.type-name,'ptr-to-',1) = 0 ) THEN DO:
   rtn_v=NO.
   RETURN.
END.

 /** it is pointer type now or array **/

rtn_v=NO.
run filter_type(output temp-type-name, types.type-name).

RUN IS_SIMPLE_TYPE(OUTPUT rtn-flag, temp-type-name).

if rtn-flag=YES THEN DO:
```

```
   return.
 END.

 temp-fun-name= passed-fun.
 FIND FIRST types WHERE  types.type-name=temp-type-name AND
    types.sys-name=passed-sys AND  types.module-name=passed-mod
    AND types.fun-name=temp-fun-name NO-LOCK NO-ERROR.

 IF NOT AVAILABLE types THEN DO:

    temp-fun-name=''.
    FIND FIRST types WHERE  types.type-name= temp-type-name and
       types.sys-name=passed-sys AND  types.module-name=passed-mod
       AND types.fun-name=temp-fun-name NO-LOCK NO-ERROR.

 END.
 IF NOT AVAILABLE types THEN
    RETURN.
 IF CAN-FIND(FIRST type-field WHERE type-field.parent-type-id= types.type-id AND
 type-field.sys-name=types.sys-name AND
    type-field.module-name= types.module-name and type-field.fun-name= types.fun-
 name) THEN DO:

    rtn_v=YES.
    return.
 END.



 END PROCEDURE.

 /***********************************************/
 procedure filter_type:
 DEF OUTPUT PARAMETER rtn-str AS CHAR.
 DEF INPUT PARAMETER passed-type AS CHAR.

 DEF VAR temp-str AS CHAR.
 DEF VAR i AS INT.

 temp-str= passed-type.


 i=index(temp-str,'array-of-',1).

 DO while index(TEMP-STR,'ARRAY-OF-',1)=1 OR INDEX(TEMP-STR,'PTR-TO-
 ',1)=1:
```

```
   IF( index(temp-str,'array-of-',1)=1) then DO:
      TEMP-STR=SUBSTRING(TEMP-STR,10).
   END.
   ELSE IF(INDEX(TEMP-STR,'PTR-TO-',1)=1) THEN DO:
      TEMP-STR=SUBSTRING(TEMP-STR,8).
   END.
END.
RTN-STR=TEMP-STR.

END PROCEDURE.
/**************************************************/
PROCEDURE is_simple_type:
DEF OUTPUT PARAMETER rtn-status as logical no-undo.
DEF INPUT  PARAMETER passed-type as char.

IF (passed-type='integer_type' or passed-type='real_type' or
   passed-type='void_type' or passed-type='enum' or passed-type='long int' or
   passed-type = 'unsinged int' ) THEN DO:
      rtn-status=YES.
END.
ELSE rtn-status=NO.

END.

/**********************************************************/
PROCEDURE IS_FUN_EXIST:
DEF OUTPUT PARAMETER rtn-num LIKE j-col.num.
DEF INPUT  PARAMETER passed-fun-id LIKE function.fun-id.
DEF INPUT  PARAMETER passed-name LIKE function.name.
DEF VAR i AS INT.

DEF BUFFER j-buf FOR j-col.

rtn-num=0.
FOR EACH j-buf WHERE j-buf.fun-id= passed-fun-id:
   if j-buf.type='FN' THEN DO:
      rtn-num=j-buf.num.
      LEAVE.
   END.
END.
IF rtn-num> 0 THEN RETURN.
i=0.
FOR EACH j-buf BY num:
   if j-buf.num > i THEN i=j-buf.num.
END.
```

```
i=i + 1.

FIND FIRST t-fun WHERE t-fun.fun-id= passed-fun-id NO-LOCK NO-ERROR.
IF NOT AVAILABLE t-fun THEN DO:
   /* consider it as sys-fun with simple-type **/
   CREATE j-buf.
   j-buf.num= i.
   j-buf.id= passed-name.
   j-buf.type='FN'.
   j-buf.fun-id= passed-fun-id.
   j-buf.is_complex=NO.
   rtn-num= i.
   RETURN.
END.

/** get function return type **/
DEF VAR is_str AS LOGICAL init no.
run is_struct(OUTPUT  is_str,t-fun.return-type, t-fun.sys-name,t-fun.module-name,
   t-fun.name).

   CREATE j-buf.
   j-buf.num= i.
   j-buf.id= passed-name.
   j-buf.type='FN'.
   j-buf.fun-id= passed-fun-id.
   j-buf.is_complex=is_str.
   rtn-num= i.



END PROCEDURE.
/**********************************************************/
PROCEDURE IS_VAR_EXIST:
DEF OUTPUT PARAMETER rtn-num LIKE j-col.num.
DEF INPUT  PARAMETER passed-name AS CHAR.
DEF INPUT  PARAMETER passed-type AS CHAR.
DEF INPUT  PARAMETER passed-fun-id LIKE function.fun-id.

DEF BUFFER fun-buf FOR function.
DEF BUFFER j-buf FOR j-col.
DEF VAR parm-list AS CHAR.
DEF VAR lclvar-list AS CHAR.
DEF VAR glb-list AS CHAR.
DEF VAR tmp-type AS CHAR.
DEF VAR temp-recid AS RECID.
```

```
FIND fun-buf WHERE fun-buf.fun-id= passed-fun-id .

run get_fun_list(BUFFER fun-buf, OUTPUT lclvar-list, OUTPUT glb-list,OUTPUT
parm-list).

tmp-type=''.
rtn-num=0.
temp-recid=0.


IF passed-type='FIELD' OR passed-type='PARAMETER' OR passed-type='VARIABLE'
THEN DO:
    IF CAN-DO(lclvar-list,passed-name) THEN
          tmp-type="LV".
    ELSE IF CAN-DO(parm-list,passed-name) THEN
          tmp-type='RG'.
    ELSE IF CAN-DO(glb-list,passed-name) THEN
          tmp-type='GV'.
END.

IF tmp-type='' THEN DO:
  BELL.
  MESSAGE 'CAN RESOLVE VARIABLE ' PASSED-NAME  VIEW-AS ALERT-
BOX.
  message 'FUNCTION'
     SKIP passed-fun-id view-as alert-box.
  RETURN.
END.
rtn-num=0.
temp-recid=0.
IF tmp-type='LV' or tmp-type='RG' THEN DO:
  FOR EACH j-buf WHERE j-buf.id= passed-name AND j-buf.type=tmp-type AND
  j-buf.fun-id = fun-buf.fun-id :
     rtn-num= j-buf.num.
     LEAVE.
  END.

END.
ELSE IF tmp-type='GV' THEN DO:
  FOR EACH j-buf WHERE j-buf.id= passed-name AND j-buf.type='GV' :
     rtn-num= j-buf.num.
     LEAVE.
  END.
END.

IF rtn-num>0 THEN
```

```
RETURN.

/* create new j-buf and find its num to be returned **/
IF tmp-type='LV' THEN DO:
   FOR EACH local-var of fun-buf:
      IF local-var.name=passed-name THEN DO:
            TEMP-RECID=RECID(LOCAL-VAR).
            LEAVE.
      END.

   END.
END.
ELSE IF tmp-type='RG' THEN DO:
   FOR EACH FUN-PARM of fun-buf:
      IF FUN-PARM.name=passed-name THEN DO:
            TEMP-RECID=RECID(FUN-PARM).
            LEAVE.
      END.
   END.
END.
ELSE IF tmp-type='GV' THEN DO:
   FOR EACH usedglb of fun-buf:
      IF usedglb.name=passed-name THEN DO:
            TEMP-RECID=RECID(usedglb).
            LEAVE.
      END.
   END.
END.
IF temp-recid<>0 then DO:
   RUN ADD-VAR-TO-J-COL(temp-recid,fun-buf.fun-id,tmp-type).
   FIND LAST j-buf WHERE j-buf.type=tmp-type AND j-buf.fun-id= fun-buf.fun-id.
   rtn-num= j-buf.num.
END.
END PROCEDURE.
/*********************************************/
PROCEDURE ADD-VAR-TO-J-COL:
DEF INPUT PARAMETER passed-recid AS RECID.
DEF INPUT PARAMETER passed-fun-id LIKE function.fun-id.
DEF INPUT PARAMETER col-type AS CHAR.

DEF BUFFER j-buf for j-col.
DEF BUFFER buf-fun for function.
DEF VAR j as int.
DEF VAR is_str AS LOGICAL.
DEF VAR var-name AS CHAR.
DEF VAR temp-fun-name AS CHAR.
```

```
DEF VAR var-typeid AS INT.


FIND buf-fun WHERE buf-fun.fun-id= passed-fun-id.

IF col-type='LV' THEN DO:
   FIND local-var WHERE RECID(local-var)=passed-recid NO-LOCK.
   var-name= local-var.name.
   var-typeid = local-var.type-id.
   temp-fun-name=buf-fun.name.
END.
IF col-type='GV' THEN DO:
   FIND usedglb WHERE RECID(usedglb)=passed-recid NO-LOCK.
   var-name   = usedglb.name.
   var-typeid = usedglb.type-id.
   temp-fun-name=''.
END.
IF col-type='RG' THEN DO:
   FIND fun-parm WHERE RECID(fun-parm)=passed-recid NO-LOCK.
   var-name   = fun-parm.name.
   var-typeid = fun-parm.type-id.
   temp-fun-name=''.
END.

   j=1.
   FOR EACH j-buf :
    j=j + 1.
   end.

   CREATE j-col.
   j-col.id= var-name.
   j-col.type=col-type.
   j-col.num = j.
   j-col.fun-id= passed-fun-id.
   is_str=NO.

   run is_struct(OUTPUT  is_str,var-typeid, buf-fun.sys-name,buf-fun.module-name,
   temp-fun-name).

   j-col.is_complex= is_str.

 END.

/************************************************/
PROCEDURE make_sys_fun:
```

```
DEF INPUT PARAMETER p-s as char.
DEF INPUT PARAMETER p-f as char.


DEF BUFFER t-buf FOR i-row.
DEF VAR i AS INT.
DEF VAR t-fun-id AS CHAR.
DEF VAR t-f AS LOGICAL.

FIND FIRST sys-fun WHERE sys-fun.sys-name= p-s AND sys-fun.name=p-f NO-
LOCK NO-ERROR.
IF NOT AVAILABLE sys-fun THEN
   t-fun-id= p-s + p-f.
ELSE
   t-fun-id= sys-fun.fun-id.

t-f=NO.

/* check if available in the matrix , if so then return **/
FOR EACH t-buf WHERE t-buf.fun-id= t-fun-id :
   t-f=YES.
   LEAVE.
END.
IF t-f=YES THEN
   RETURN t-fun-id.

i=1.
FOR EACH t-buf BY t-buf.num:
   if t-buf.num > i THEN
      i= t-buf.num + 1.
END.

CREATE t-buf.
t-buf.proc-name= p-f.
t-buf.num= i.
t-buf.fun-id= t-fun-id.

RETURN t-fun-id.
END PROCEDURE.
/**********************************************/
PROCEDURE get_fun_list:
DEF PARAMETER BUFFER fun-buf FOR function.
DEF OUTPUT PARAMETER r-lcl AS CHAR.
DEF OUTPUT PARAMETER r-glb AS CHAR.
DEF OUTPUT PARAMETER r-prm AS CHAR.
```

```
r-lcl=''.
r-glb=''.
r-prm=''.

FOR EACH fun-parm OF fun-buf NO-LOCK:
   r-prm= r-prm + fun-parm.name + ','.
END.
FOR EACH local-var of fun-buf NO-LOCK:
   r-lcl= r-lcl + local-var.name + ','.
END.
FOR EACH usedglb OF fun-buf NO-LOCK:
   r-glb= r-glb + usedglb.name + ",".
END.


END PROCEDURE.
/*********************************/
PROCEDURE get_fun_id:
DEF OUTPUT PARAMETER r-id LIKE function.fun-id.
DEF INPUT PARAMETER p-s AS CHAR.
DEF INPUT PARAMETER p-m AS CHAR.
DEF INPUT PARAMETER p-f AS CHAR.

DEF BUFFER t-fun FOR function.
/** check same module **/
FIND FIRST t-fun WHERE t-fun.sys-name= p-s AND t-fun.module-name=p-m AND
   t-fun.name = p-f NO-LOCK NO-ERROR.
IF NOT AVAILABLE t-fun THEN
   FIND FIRST t-fun WHERE t-fun.sys-name= p-s AND t-fun.name = p-f NO-LOCK
NO-ERROR.

IF NOT AVAILABLE t-fun THEN
   r-id= p-s + p-m + p-f.
ELSE
   r-id= t-fun.fun-id.


END PROCEDURE.
/***************************************************************/
PROCEDURE SEND-TO-EXCEL:

DEF VAR sheet1 AS INT.
DEF VAR DDFRAME AS HANDLE .
DEF VAR loc-str AS CHAR.
DEF VAR r-str AS CHAR.
DEF VAR c-str AS CHAR.
```

```
DEF VAR irow AS INT.
DEF VAR jcol AS INT.
DEF FRAME dde-frame
""
with center.
view frame dde-frame.
hide frame dde-frame.
DDFRAME= FRAME dde-frame:HANDLE.

DDE INITIATE SHEET1 FRAME DDFRAME APPLICATION "EXCEL" TOPIC
"SHEET1".


FOR EACH j-col BY j-col.num:

   FIND FIRST i-row WHERE i-row.fun-id= j-col.fun-id.

   r-str="C" + STRING(j-col.num + 1) .
   loc-str= 'R1' + r-str.
   DDE SEND SHEET1 SOURCE i-row.proc-name  item loc-str.
   loc-str= 'R2' + r-str.
   DDE SEND SHEET1 SOURCE j-col.type  item loc-str.
   loc-str= 'R3' + r-str.
   DDE SEND SHEET1 SOURCE j-col.id    item loc-str.
   loc-str= 'R4' + r-str.
   DDE SEND SHEET1 SOURCE (IF j-col.is_complex  THEN "COMPLEX" ELSE
"SIMPLE")  item loc-str.
END.

irow=5.
FOR EACH i-row BY i-row.num:
   r-str="R" + STRING(irow) .

   loc-str= r-str + 'C1'.
   DDE SEND SHEET1 SOURCE i-row.proc-name  item loc-str.

   jcol=2.
   FOR EACH j-col BY j-col.num:
      FIND FIRST i-j-link WHERE i-j-link.i-num= i-row.num AND
         i-j-link.j-num= j-col.num NO-ERROR.

      loc-str='R' + string(irow) + 'C' + STRING(jcol).
      IF AVAILABLE i-j-link THEN
         DDE SEND SHEET1 SOURCE STRING(i-j-link.link-v)  item loc-str.
      ELSE
         DDE SEND SHEET1 SOURCE STRING(0)  item loc-str.
```

```
      jcol= jcol + 1.
   END.
      irow= irow + 1.
END.

irow = irow + 3.
JCOL=2.
NUM-OF-FUN=0.
FOR EACH I-MAT:
   NUM-OF-FUN=NUM-OF-FUN + 1.
   r-str="R" + STRING(IROW)  .

   loc-str= r-str + 'C' + STRING(JCOL).
   DDE SEND SHEET1 SOURCE i-MAT.NAME  item loc-str.
   JCOL= JCOL + 1.
end.

IROW=IROW  + 1.
TOT-CP=0.
AVG-CP=0.

FOR EACH i-mat:

   r-str="R" + STRING(IROW)  .
   loc-str= r-str + 'C1' .
   DDE SEND SHEET1 SOURCE i-MAT.NAME  item loc-str.
   JCOL=2.

   FOR EACH j-mat :
      r-str="R" + STRING(IROW)  .
      loc-str= r-str + 'C' + STRING(JCOL) .

      FIND FIRST i-j-mat WHERE i-j-mat.i-num= i-mat.num
                AND i-j-mat.j-num= j-mat.num.

      DDE SEND SHEET1 SOURCE  string(i-J-MAT.I-J-VALUE)  item  loc-str.
      JCOL=JCOL + 1.
      IF i-mat.num <> j-mat.num THEN
         tot-cp= tot-cp + i-j-mat.i-j-value.

   END.
   IROW= IROW + 1.
END.

AVG-CP= TOT-CP / NUM-OF-FUN.
```

```
IROW=IROW + 1.

r-str="R" + STRING(IROW) .
loc-str= r-str + 'C1' .
DDE SEND SHEET1 SOURCE  "SYSTEM COUPLING"  item  loc-str.
loc-str= r-str + 'C3' .
DDE SEND SHEET1 SOURCE  STRING(TOT-CP)  item  loc-str.

IROW=IROW + 1.
r-str="R" + STRING(IROW) .
loc-str= r-str + 'C1' .
DDE SEND SHEET1 SOURCE  "AVERAGE COUPLING"  item  loc-str.
loc-str= r-str + 'C3' .
DDE SEND SHEET1 SOURCE  STRING(AVG-CP)  item  loc-str.


END.
```

Appendix D

This appendix contains six figures that are the listings of three C programs.

All the three programs achieve the same functionality, which is sorting an array of integers, however the implementation of each program is different.

Listing of program sort 1.

Sort 1

This program sorts an array of integers. the array is global variable and used by function main only. Main function passes this array to function

sort as a parameter.

```
int vec[10]={5,3,1,14,5,7,8,9,99,22};

void swap(int *p, int *q ){
int tmp;
tmp=*p;
*p= *q;
*q= tmp;
};

void sort(int *vec ){
int i,j, k, k1;
    for (j=0;j<10;j++){
            for(i=0;i< 9; i++){
                    if(vec[i] >  vec[i+1]){
                            k=vec[i]; k1=vec[i+1];
                            swap(&k,&k1);
                            vec[i]=k; vec[i+1]=k1;
                    }
            }
    }
}
```

```
main(){
        int i;
    vec[0]=10;
        sort(&vec[0]);
        printf("\n");
}
```

Listing of sort 11.

```
int vec[10]={5,3,1,14,5,7,8,9,99,22};

int is_greater(int x, y) {
        if(x>y) return 1;
else return 0;
}

void swap(int *p, int *q ){
int tmp;
tmp=*p;
*p= *q;
*q= tmp;
};

void sort(int *vec ){
int i,j, k, k1;
        for (j=0;j<10;j++){
                for(i=0;i< 9; i++){
                        if(vec[i] >  vec[i+1]){
                                k=vec[i]; k1=vec[i+1];
                                swap(&k,&k1);
                                vec[i]=k; vec[i+1]=k1;
                        }
                }
        }
}

main(){
int i;
vec[0]=10;
sort(&vec[0]);
printf("\n");
}
```

List of program sort 2.

Sort 2

This program sorts an array of integers. the array is local variable

of main function , which in turn passes it to function sort as parameter.

```
void swap(int *p, int *q ){
int tmp;
    tmp=*p;
*p= *q;
*q= tmp;
};

void sort(int *vec){
int i,j, k, k1;
    for (j=0;j<10;j++){
            for(i=0;i< 9; i++){
                    if(vec[i] >  vec[i+1]){
                            k=vec[i]; k1=vec[i+1];
                            swap(&k,&k1);
                            vec[i]=k; vec[i+1]=k1;
                    }
            }
    }
}
main(){
int i;
int vec[10]={5,3,1,14,5,7,8,9,99,22};

vec[0]=10;
sort(&vec[0] );
printf("\n");
}
```

Listing of program sort 21.

```
int is_greater(int x, int y) {
    if(x>y) return 1;
    else return 0;
};

void swap(int *p, int *q ){
int tmp;
    tmp=*p;
*p= *q;
*q= tmp;
};

void sort(int *vec){
int i,j, k, k1;
    for (j=0;j<10;j++){
        for(i=0;i< 9; i++){
            if(vec[i] > vec[i+1]){
                k=vec[i]; k1=vec[i+1];
                swap(&k,&k1);
                vec[i]=k; vec[i+1]=k1;
            }
        }
    }
}
main(){
int i;
int vec[10]={5,3,1,14,5,7,8,9,99,22};

vec[0]=10;
sort(&vec[0] );
printf("\n");
}
```

Listing of program sort 3.

Sort 3

This program sorts an array of integers. The array is global variable

used in main and sort functions

```
int vec[10]={5,3,1,14,5,7,8,9,99,22};

void swap(int *p, int *q ){
int tmp;
tmp=*p;
*p= *q;
*q= tmp;
};

void sort(){
int j, k, k1, i;
    for (j=0;j<10;j++){
            for(i=0;i< 9; i++){
                    if(vec[i] >  vec[i+1]){
                            k=vec[i]; k1=vec[i+1];
                            swap(&k,&k1);
                            vec[i]=k; vec[i+1]=k1;
                    }
            }
    }
}

main(){

int i;

vec[0]=10;
sort();
printf("\n");
}
```

Listing of program sort 31.

```
int vec[10]={5,3,1,14,5,7,8,9,99,22};

int is_greater(int x, y) {
        if(x>y) return 1;
else return 0;
}

void swap(int *p, int *q ){
int tmp;
tmp=*p;
*p= *q;
*q= tmp;
};

void sort(){
int j, k, k1, i;
        for (j=0;j<10;j++){
                for(i=0;i< 9; i++){
                        if(vec[i] > vec[i+1]){
                                k=vec[i]; k1=vec[i+1];
                                swap(&k,&k1);
                                vec[i]=k; vec[i+1]=k1;
                        }
                }
        }
}

main(){

int i;

vec[0]=10;
sort();
printf("\n");
}
```

Appendix E

User Manual

The following describes the system requirements and the procedures that a user must perform to obtain system couplings.

System Requirements

- Linux operating system.
- Windows 95 operating system.
- Progress RDBMS.

Procedures:

Create a text file and name it "sysfile". Edit the sysfile and type the name of the system under consideration. The text file should created under linux operating system.

At linux shell prompt type *compile module-name* without the extension ".c".

The compile command is a shell script that performs the following:

1. Invokes the preprocessor of C compiler to process the marcos and the include files.
2. It passes the output of the preprocessor into the awk utilities which further filters down any unwanted debugging lines introduced by the preprocessor.

3. The output of awk utility is then passed on to the special purpose parser to produce the system description of the module under consideration.
4. The special purpose parser produces the text files that are uploaded by database utilities.
5. Copy the text files produced by special purpose parser into a dos/windows directory.

- Switch the environment to windows 95 operating system.
- Run Progress database system utilities and upload the text files produced in previous stage.
- Run the following programs
    1. aliases.p to create a parameter aliases of the system.
    2. sysfun.p to create system function table.
- Run/Develop and module to calculate the desired coupling method.

Appendix F

This appendix contains the listing of utility program writtne in Progress 4GL. The program scans the function calls in the system and creates records for any system function found.

Listing of the program.

/** this procedure identifies the function that are not define in the system

and assume them to system calls, hence save the in sys-function-table **/

```
DEF TEMP-TABLE tf   FIELD fun-id LIKE sys-fun.fun-id
                    FIELD p    as int
                    index fun-id fun-id.

def buffer c-fun for function.
def var s-name as char.
DEF VAR parm-c AS INT.
DEF VAR i AS INT.
DEF VAR j AS INT.

s-name='S4'.

FOR EACH function WHERE function.sys-name = s-name NO-LOCK:

    FOR EACH fun-call of function NO-LOCK:

        find first c-fun where c-fun.name= fun-call.callee
        AND c-fun.sys-name=function.sys-name  NO-ERROR .
        if AVAILABLE c-fun THEN
            NEXT.

        FIND FIRST sys-fun where sys-fun.name=callee  AND
        sys-fun.sys-name = function.sys-name NO-LOCK
        NO-ERROR.
        IF NOT AVAILABLE sys-fun THEN DO:
            create sys-fun.
```

```
            sys-fun.sys-name=function.sys-name.
            sys-fun.name=callee.
            sys-fun.fun-id= sys-fun.sys-name +
                              sys-fun.name.

            DISP sys-fun.name "CREATED". pause 0.

        END.

    END.
END.

FOR EACH sys-fun WHERE sys-fun.sys-name = s-name NO-LOCK:
    FOR EACH sf-parm of sys-fun:
        DELETE sf-parm.
    END.
END.

FOR EACH function WHERE function.sys-name = s-name NO-LOCK:

    FOR EACH fun-call of function :

        FIND first sys-fun where sys-fun.name= callee and
        sys-fun.sys-name= function.sys-name NO-LOCK
        NO-ERROR.
        IF NOT AVAILABLE sys-fun  THEN
            NEXT.

        FIND FIRST tf WHERE tf.fun-id= sys-fun.fun-id
        NO-ERROR.
        IF NOT AVAILABLE tf THEN DO:
            CREATE tf. tf.fun-id= sys-fun.fun-id.
        END.

        i=0.
        FOR EACH passed-arg where passed-arg.call-id=
          fun-call.call-id NO-LOCK:
            i=i + 1.
        END.

        if i=0 THEN NEXT.

        if i> tf.p THEN tf.p= i.
    END.
END.

FOR EACH tf :
```

```
    DO i=1 to tf.p:
        create sf-parm.
        sf-parm.fun-id= tf.fun-id.
        sf-parm.position= i.
        sf-parm.name = "P" + STRING(i).
    END.
END.
```

APPENDIX G

	add_complex_t	dump_data	dump_data	xtend_var_lis	get_exp_vari	ess_single_p	complex_t	complex_t	complex_t	complex_t	complex_t	complex_t	add_type	add_type	add_type	break_comp	break_comp	break_comp
	GV	GV	GV	GV	GV	GV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV
	s_data_type	lobal_var_lis	fun_vec	temp_var_list	st_cond_typ	rtn_para_list	temp_name	t_num	temp_field	f_type	type_to_add	f	type_num	w_type_to_a	t	t1	t0	t
	COMPLEX	COMPLEX	COMPLEX	COMPLEX	SIMPLE	COMPLEX	SIMPLE	SIMPLE	COMPLEX	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE
add_call_to_list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
add_complex_type	22	0	0	0	0	0	1	16	0	4	0	0	9	0	3	0	0	0
add_field_to_list	4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
add_para_to_list	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0
add_simple_type	28	0	0	0	0	0	0	4	0	0	0	0	25	7	0	0	0	0
add_str_to_exp_var	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
add_type	0	0	0	0	0	0	0	0	0	0	0	0	18	7	15	0	0	0
add_var_to_list	0	4	8	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
brace_name	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
break_comp	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	6	4	10
dump_data	46	12	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
extend_var_list	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0
filter_comp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
get_code_name	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
get_component_na	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
get_exp_var1	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	3
get_exp_var_list	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0
get_last_call_id	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
get_system_name	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
get_type_id	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
initialize_data_types	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
is_complex	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
is_expression	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
is_single_operand	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
make_field	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
make_fun_call	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
make_parameter_no	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
make_var	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mk_parm	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
modify_var	0	0	30	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
print_complex_type	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
print_fun_calls	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
print_parameter_list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
print_simple_type	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
print_var_list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
process_array_poin	0	0	0	0	0	0	0	2	0	0	0	0	1	0	2	0	0	0
process_calls_parm	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
process_cond_exp	0	0	0	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0
process_enum	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
process_one_call_p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
process_read_write	0	0	0	84	0	0	0	0	0	0	0	0	0	0	0	0	0	0
process_single_par	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0
quote_str	6	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
return_name	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0
return_simple_type	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
return_type_name	8	0	0	0	0	0	0	3	0	0	0	0	9	0	0	0	0	0
return_type_num	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
single_fun_id	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
var_exist	0	0	18	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
write_last_call_id	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
copy_node	0	0	0	0	0	0	0	2	0	0	0	0	1	0	2	0	0	0
get_decls	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nreverse	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table G.1 The D-Matrix of FCMS.

getdecls	et_last_call_i	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	dump_data	xtend_var_lis	xtend_var_lis
FN	FN	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	LV
getdecls	et_last_call_i	temp_field1	temp_field2	glob	t	temp_field3	temp_field4	var	fun_id	perm	fc	p_list	temp_field5	temp_field6	temp_field7	f	last_call_id	l	type_num
SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	COMPLEX	SIMPLE	COMPLEX	COMPLEX	COMPLEX	SIMPLE	SIMPLE	SIMPLE	COMPLEX	SIMPLE	SIMPLE	SIMPLE
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
5	5	3	7	0	3	4	8	42	0	0	0	8	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	3	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	3	3	0	0	3	8	20	0	0	0	4	1	0	0	0	0	0	0
0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

xtend_var_lis	xtend_var_lis	xtend_var_lis	filter_comp	filter_comp	filter_comp	get_exp_varl	get_exp_varl	get_exp_varl	st_exp_var_list	_exp_var_li	mk_parm	mk_parm	mk_parm	mk_parm	modify_var	rint_fun_call	print_var_list	ess_array_pc	ess_array_pc
LV	LV	LV	LV	LV	LV	LV	LV	LV	LV	GV	LV	LV	LV	LV	LV	RG	RG	LV	LV
t	comp_name	wfield_name	t	t_str	rtn_str	l	r	m	t	bal_num_of_	p	temp_name	t_uid	t_v	temp	p_calls	l	tn	st
COMPLEX	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	COMPLEX	SIMPLE	SIMPLE	SIMPLE	SIMPLE	COMPLEX	COMPLEX	COMPLEX	COMPLEX	SIMPLE	SIMPLE
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	4	2	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	9	15	3	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	8	5	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	3	6	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	3	6	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3	0	0	0	0	0	0	0	4	0	0	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	3	1	1	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4	4	0	0	0	0	4	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2	4	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ess_array_pcess_	array_pccess_	calls_p	ss_one_call	ss_read_writ	ess_single_p	ess_single_p	ess_single_p	ess_single_p	ess_single_p	rm_simple_t	rm_simple_type
LV	LV	LV	LV	RG	LV	GV	GV	GV	LV	RG	RG
ll	tm	p	p	p_var	l	mp_para_nan	mp_para_typ	p_para_exp_	tmp_para	origin	l
SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	SIMPLE	COMPLEX	SIMPLE	SIMPLE
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	12	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	15	0	0	0	0	0	0
0	0	0	0	0	12	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	3	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	3	0	4	0	0	0	0	0	0
0	0	0	0	0	9	0	0	0	0	0	0
0	0	0	1	0	39	10	10	0	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	9	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

	add_call_to_lis	d_complex_ty	d_field_to_lis	d_para_to_lis	d_simple_ty	l_str_to_exp_	add_type	dd_var_to_lis	brace_name	break_comp	dump_data
add_call_to_list	0	0	0	0	0	0	0	0	0	0	0
add_complex_type	0	0.326146438	0.019669902	0	0.172530337	0	0.08394333	0	0	0	0.110843374
add_field_to_list	0	0.216368924	0.02617366	0	0.162525966	0	0	0	0	0	0.221686748
add_para_to_list	0	0	0	0.333333333	0	0	0	0	0	0	0
add_simple_type	0	0.139546596	0.011950439	0	0.288002053	0	0.156512606	0	0	0	0.114103473
add_str_to_exp_va	0	0	0	0	0	0	0	0	0	0	0
add_type	0	0.115422078	0	0	0.266071429	0	0.471753247	0	0	0	0
add_var_to_list	0	0	0	0	0	0	0	0.182109239	0	0	0.154000167
brace_name	0	0	0	0	0	0	0	0	0	0	0
break_comp	0	0	0	0	0	0	0	0	0	0.497125424	0
dump_data	0	0.029309546	0.005329008	0	0.037303058	0	0	0.034798115	0	0	0.480212099
extend_var_list	0	0	0	0	0	0	0	0.088345865	0	0.012110727	0
filter_comp	0	0	0	0	0	0	0	0	0	0.066666667	0
get_code_name	0	0	0	0	0.222222222	0	0	0	0	0	0
et_component_nam	0	0	0	0	0	0	0	0.013888889	0	0.050108933	0
get_exp_var1	0	0	0	0	0	0	0	0	0	0.088811189	0
get_exp_var_list	0	0	0	0	0	0	0	0.051264525	0	0	0
get_last_call_id	0	0	0	0	0	0	0	0	0	0	0.5
get_system_name	0	0	0	0	0	0	0	0	0	0	0
get_type_id	0	0	0	0	0	0	0	0.143716578	0	0	0.563502675
nitialize_data_type	0	0.132530121	0.024096386	0	0.168674699	0	0	0	0	0	0.277108435
is_complex	0	0	0	0	0	0	0	0	0	0	0
is_expression	0	0	0	0	0	0	0	0	0	0.090909091	0
is_single_operand	0	0	0	0	0	0	0	0	0	0.056818182	0
make_field	0	0.525862069	0.017241379	0	0.068965517	0	0	0	0	0	0
make_fun_call	0	0	0	0	0	0	0	0	0	0	0
ake_parameter_no	0	0	0	0	0	0	0	0	0	0	0
make_var	0	0	0	0	0	0	0	0.041151987	0	0.006302521	0.044642857
mk_parm	0	0	0	0	0	0	0	0.090337785	0	0	0.301649648
modify_var	0	0	0	0	0	0	0	0.074567292	0	0	0.265339968
print_complex_typ	0	0	0	0	0	0	0	0	0	0	0
print_fun_calls	0	0	0	0	0	0	0	0	0	0	0
print_parameter_lis	0	0	0	0	0	0	0	0	0	0	0
print_simple_type	0	0.132530121	0.024096386	0	0.168674699	0	0	0	0	0	0.277108435
print_var_list	0	0	0	0	0	0	0	0	0	0	0
rocess_array_point	0	0.101268846	0.004597701	0	0.148549535	0	0.10995671	0	0	0	0
process_calls_parm	0	0	0	0	0	0	0	0	0	0	0
process_cond_exp	0	0	0	0	0	0	0	0.07518797	0	0	0
process_enum	0	0	0	0	0	0	0	0	0	0	0
ocess_one_call_pa	0	0	0	0	0	0	0	0	0	0	0
ocess_read_write_	0	0	0	0	0	0	0	0.065789474	0	0	0
rocess_single_par	0	0	0	0.04494382	0	0	0	0	0	0	0
quote_str	0	0.015903615	0.002891566	0	0.020240964	0	0	0.014152766	0	0	0.532749648
return_name	0	0	0	0	0	0	0	0	0	0.124183006	0.055555556
return_simple_type	0	0	0	0	0.111111111	0	0	0	0	0	0
return_type_name	0	0.173962073	0.012879103	0	0.260925476	0	0.111801243	0	0	0	0.096385543
return_type_num	0	0.132530121	0.024096386	0	0.168674699	0	0	0	0	0	0.277108435
single_fun_id	0	0	0	0	0	0	0	0	0	0	0
var_exist	0	0	0	0	0	0	0	0.05085849	0	0	0.286567165
write_last_call_id	0	0	0	0	0	0	0	0.029411765	0	0	0.617647061
copy_node	0	0.253172115	0.011494253	0	0.112114578	0	0.274891775	0	0	0	0
getdecls	0	0	0	0	0	0	0	0	0	0	0.5
nreverse	0	0	0	0	0	0	0	0	0	0	0.5

SYSTEM COUPLING 30.26153614

Table G.2 The CC-Matrix of the FCMS.

	extend_var_lis	filter_comp	et_code_nam	component_r	get_exp_var1	et_exp_var_li	et_last_call_i	t_system_na	get_type_id	ialize_data_ty	is_complex
add_call_to_list	0	0	0	0	0	0	0	0	0	0	0
add_complex_type	0	0	0	0	0	0	0	0	0	0.057831326	0
add_field_to_list	0	0	0	0	0	0	0	0	0	0.115662651	0
add_para_to_list	0	0	0	0	0	0	0	0	0	0	0
add_simple_type	0	0	0.003267974	0	0	0	0	0	0	0.059532247	0
add_str_to_exp_va	0	0	0	0	0	0	0	0	0	0	0
add_type	0	0	0	0	0	0	0	0	0	0	0
add_var_to_list	0.191729323	0	0	0.002659575	0	0.047992321	0	0	0.024462396	0	0
brace_name	0	0	0	0	0	0	0	0	0	0	0
break_comp	0.044117647	0.011904762	0	0.016106443	0.158591409	0	0	0	0	0	0
dump_data	0	0	0	0	0	0	0.012019231	0	0.02167318	0.03197405	0
extend_var_list	0.351496444	0	0	0.03171857	0.018119748	0.066342327	0	0	0	0	0
filter_comp	0	0.285294118	0	0.266666667	0.023529412	0.001960784	0	0	0	0	0
get_code_name	0	0	0.055555556	0	0	0	0	0	0	0	0
et_component_nan	0.359477124	0.148148148	0	0.300108932	0	0	0	0	0	0	0
get_exp_var1	0.036964286	0.002352941	0	0	0.441842937	0.002352941	0	0	0	0	0
get_exp_var_list	0.153793575	0.000222816	0	0	0.002673797	0.17625613	0	0	0	0	0
get_last_call_id	0	0	0	0	0	0	0.5	0	0	0	0
get_system_name	0	0	0	0	0	0	0	0	0	0	0
get_type_id	0	0	0	0	0	0	0	0	0.211898396	0	0
nitialize_data_type	0	0	0	0	0	0	0	0	0	0.144578314	0
is_complex	0	0	0	0	0	0	0	0	0	0	0
is_expression	0.013392857	0.004901961	0	0	0.249489114	0.004901961	0	0	0	0	0
is_single_operand	0.016741071	0.004901961	0	0	0.280110056	0.004901961	0	0	0	0	0
make_field	0	0	0	0	0	0	0	0	0	0	0
make_fun_call	0	0	0	0	0	0	0	0	0	0	0
ake_parameter_no	0	0.027777778	0	0	0	0	0	0	0	0	0
make_var	0.252132432	0	0	0.02862395	0	0.111827187	0	0	0	0	0
mk_parm	0	0	0	0	0	0	0	0	0	0	0
modify_var	0.050125313	0	0	0	0	0.069931899	0	0	0	0	0
print_complex_typ	0	0	0	0	0	0	0	0	0	0	0
print_fun_calls	0	0	0	0	0	0	0	0	0	0	0
print_parameter_lis	0	0	0	0	0	0	0	0	0	0	0
print_simple_type	0	0	0	0	0	0	0	0	0	0.144578314	0
print_var_list	0	0	0	0	0	0	0	0	0	0	0
rocess_array_point	0	0	0.025925926	0	0	0	0	0	0	0	0
process_calls_parm	0	0	0	0	0	0	0	0	0	0	0
process_cond_exp	0.22556391	0	0	0	0	0.112781955	0	0	0	0	0
process_enum	0	0	0	0	0	0	0	0	0	0	0
ocess_one_call_pa	0	0.004901961	0	0	0.05882353	0.004901961	0	0	0	0	0
ocess_read_write_	0.197368422	0.000919118	0	0	0.011029412	0.104643651	0	0	0	0	0
rocess_single_par	0	0.013659396	0	0	0.051553206	0.031479863	0	0	0	0	0
quote_str	0	0	0	0	0	0	0	0	0.011764706	0.017349398	0
return_name	0.137254902	0.003267974	0	0.013071895	0.039215686	0.003267974	0	0	0	0	0
return_simple_typ	0	0	0.027777778	0	0	0	0	0	0	0	0
return_type_name	0	0	0.007246377	0	0	0	0	0	0	0.050288109	0
return_type_num	0	0	0	0	0	0	0	0	0	0.144578314	0
single_fun_id	0	0	0	0	0	0	0	0	0	0	0
var_exist	0.045112782	0	0	0	0	0.073691397	0	0	0	0	0
write_last_call_id	0	0	0	0	0	0	0	0	0.029411765	0	0
copy_node	0	0	0	0	0	0	0	0	0	0	0
getdecls	0	0	0	0	0	0	0	0	0	0	0
nreverse	0	0	0	0	0	0	0	0	0	0	0

SYSTEM COUPLING

	ais_expressior	single_opera	make_field	nake_fun_call	_parameter_	make_var	mk_parm	modify_var	it_complex_t	print_fun_call	it_parameter_
add_call_to_list	0	0	0	0	0	0	0	0	0	0	0
add_complex_type	0	0	0.019122257	0	0	0	0	0	0	0	0
add_field_to_list	0	0	0.006896552	0	0	0	0	0	0	0	0
add_para_to_list	0	0	0	0	0	0	0	0	0	0	0
add_simple_type	0	0	0.002028398	0	0	0	0	0	0	0	0
add_str_to_exp_va	0	0	0	0	0	0	0	0	0	0	0
add_type	0	0	0	0	0	0	0	0	0	0	0
add_var_to_list	0	0	0	0	0	0.024516077	0.03651953	0.085673059	0	0	0
brace_name	0	0	0	0	0	0	0	0	0	0	0
break_comp	0.097402598	0.048701299	0	0	0	0.006302521	0	0	0	0	0
dump_data	0	0	0	0	0	0.006009615	0.027554535	0.068886338	0	0	0
extend_var_list	0.003939076	0.003939076	0	0	0	0.069212825	0	0.026536931	0	0	0
filter_comp	0.029411765	0.023529412	0	0	0.016666667	0	0	0	0	0	0
get_code_name	0	0	0	0	0	0	0	0	0	0	0
et_component_nan	0	0	0	0	0	0.089052288	0	0	0	0	0
get_exp_var1	0.149693468	0.134452827	0	0	0	0	0	0	0	0	0
get_exp_var_list	0.003342246	0.002673797	0	0	0	0.071162755	0	0.085825512	0	0	0
get_last_call_id	0	0	0	0	0	0	0	0	0	0	0
get_system_name	0	0	0	0	0	0	0	0	0	0	0
get_type_id	0	0	0	0	0	0	0	0	0	0	0
nitialize_data_type	0	0	0	0	0	0	0	0	0	0	0
is_complex	0	0	0	0	0	0	0	0	0	0	0
is_expression	0.189682009	0.1477034	0	0	0	0	0	0	0	0	0
is_single_operand	0.18462925	0.152877913	0	0	0	0	0	0	0	0	0
make_field	0	0	0.267241379	0	0	0	0	0	0	0	0
make_fun_call	0	0	0	0	0	0	0	0	0	0	0
ake_parameter_no	0	0	0	0	0.138888889	0	0	0	0	0	0
make_var	0	0	0	0	0	0.16767672	0.035714286	0.051080081	0	0	0
mk_parm	0	0	0	0	0	0.052631579	0.240769835	0.141398272	0	0	0
modify_var	0	0	0	0	0	0.026485968	0.049751244	0.209804739	0	0	0
print_complex_typ	0	0	0	0	0	0	0	0	0	0	0
print_fun_calls	0	0	0	0	0	0	0	0	0	0	0
print_parameter_lis	0	0	0	0	0	0	0	0	0	0	0
print_simple_type	0	0	0	0	0	0	0	0	0	0	0
print_var_list	0	0	0	0	0	0	0	0	0	0	0
rocess_array_point	0	0	0.004597701	0	0	0	0	0	0	0	0
process_calls_parn	0	0	0	0	0	0	0	0	0	0	0
process_cond_exp	0	0	0	0	0	0.045112782	0	0.045112782	0	0	0
process_enum	0	0	0	0	0	0	0	0	0	0	0
ocess_one_call_pa	0.073529412	0.05882353	0	0	0	0	0	0	0	0	0
ocess_read_write_	0.013786765	0.011029412	0	0	0	0.039473684	0	0.043505943	0	0	0
rocess_single_par	0.064441507	0.051553206	0	0	0.028089888	0	0	0.02174701	0	0	0
quote_str	0	0	0	0	0	0.0075	0.00358209	0.008955224	0	0	0
return_name	0.049019608	0.039215686	0	0	0	0.019607843	0.055555556	0	0	0	0
return_simple_typ	0	0	0	0	0	0	0	0	0	0	0
return_type_name	0	0	0.004497751	0	0	0	0	0	0	0	0
return_type_num	0	0	0	0	0	0	0	0	0	0	0
single_fun_id	0	0	0	0	0	0	0	0	0	0	0
var_exist	0	0	0	0	0	0.023837371	0.053731344	0.175370031	0	0	0
write_last_call_id	0	0	0	0	0	0	0	0	0	0	0
copy_node	0	0	0.011494253	0	0	0	0	0	0	0	0
getdecls	0	0	0	0	0	0	0	0	0	0	0
nreverse	0	0	0	0	0	0	0	0	0	0	0

SYSTEM COUPLING

	aint_simple_ty	print_var_list	ess_array_po	cess_calls_p	ocess_cond_	process_enum	ss_one_call_	ss_read_writ	ess_single_p	quote_str	return_name
add_call_to_list	0	0	0	0	0	0	0	0	0	0	0
add_complex_type	0.053012048	0	0.027618776	0	0	0	0	0	0	0.014457831	0
add_field_to_list	0.106024097	0	0.013793103	0	0	0	0	0	0	0.028915663	0
add_para_to_list	0	0	0	0	0	0	0	0	0.666666667	0	0
add_simple_type	0.054571226	0	0.03276828	0	0	0	0	0	0	0.014883062	0
add_str_to_exp_va	0	0	0	0	0	0	0	0	0	0	0
add_type	0	0	0.041233766	0	0	0	0	0	0	0	0
add_var_to_list	0	0	0	0	0.06718925	0	0	0.134378499	0	0.015056134	0
brace_name	0	0	0	0	0	0	0	0	0	0	0
break_comp	0	0	0	0	0	0	0	0	0	0	0.119747899
dump_data	0.029309546	0	0	0	0	0	0	0	0	0.128064819	0.007211538
extend_var_list	0	0	0	0	0.092879257	0	0	0.185758515	0	0	0.03633218
filter_comp	0	0	0	0	0	0	0.007843137	0.017647059	0.243137255	0	0.017647059
get_code_name	0	0	0.388888889	0	0	0	0	0	0	0	0
et_component_nam	0	0	0	0	0	0	0	0	0	0	0.039215686
get_exp_var1	0	0	0	0	0	0	0.009411765	0.021176471	0.091764706	0	0.021176471
get_exp_var_list	0	0	0	0	0.107655502	0	0.000891266	0.22831342	0.063675177	0	0.002005348
get_last_call_id	0	0	0	0	0	0	0	0	0	0	0
get_system_name	0	0	0	0	0	0	0	0	0	0	0
get_type_id	0	0	0	0	0	0	0	0	0	0.073529412	0
nitialize_data_type	0.132530121	0	0	0	0	0	0	0	0	0.036144578	0
is_complex	0	0	0	0	0	0	0	0	0	0	0
is_expression	0	0	0	0	0	0	0.019607843	0.044117647	0.191176471	0	0.044117647
is_single_operand	0	0	0	0	0	0	0.019607843	0.044117647	0.191176471	0	0.044117647
make_field	0	0	0.034482759	0	0	0	0	0	0	0	0
make_fun_call	0	0	0	0	0	0	0	0	0	0	0
ake_parameter_no	0	0	0	0	0	0	0	0	0.833333333	0	0
make_var	0	0	0	0	0.067669173	0	0	0.135338346	0	0.013392857	0.018907563
mk_parm	0	0	0	0	0	0	0	0	0	0.009426552	0.078947368
modify_var	0	0	0	0	0.035087719	0	0	0.077343897	0.035842294	0.008291874	0
print_complex_type	0	0	0	0	0	0	0	0	0	0	0
print_fun_calls	0	0	0	0	0	0	0	0	0	0	0
print_parameter_lis	0	0	0	0	0	0	0	0	0	0	0
print_simple_type	0.132530121	0	0	0	0	0	0	0	0	0.036144578	0
print_var_list	0	0	0	0	0	0	0	0	0	0	0
rocess_array_point	0	0	0.303856297	0	0	0	0	0	0	0	0
process_calls_parm	0	0	0	0.75	0	0	0.25	0	0	0	0
process_cond_exp	0	0	0	0	0.157894737	0	0	0.315789474	0	0	0
process_enum	0	0	0	0	0	0	0	0	0	0	0
ocess_one_call_pa	0	0	0	0.09375	0	0	0.275857843	0.044117647	0.266176471	0	0.044117647
ocess_read_write_	0	0	0	0	0.138157895	0	0.003676471	0.287612043	0.050966556	0	0.008272059
rocess_single_par	0	0	0	0	0	0	0.023925975	0.054975162	0.550971867	0	0.038664904
quote_str	0.015903615	0	0	0	0	0	0	0	0	0.304605233	0.012857143
return_name	0	0	0	0	0	0	0.013071896	0.029411765	0.127450981	0.023809524	0.259103641
return_simple_type	0	0	0.444444445	0	0	0	0	0	0	0	0
return_type_name	0.046097433	0	0.06593132	0	0	0	0	0	0	0.012572027	0
return_type_num	0.132530121	0	0	0	0	0	0	0	0	0.036144578	0
single_fun_id	0	0	0	0	0	0	0	0	0	0	0
var_exist	0	0	0	0	0.031578947	0	0	0.076061121	0.064516129	0.008955224	0
write_last_call_id	0	0	0	0	0	0	0	0	0	0.294117648	0
copy_node	0	0	0.055937039	0	0	0	0.1	0	0.033333333	0	0
getdecls	0	0	0	0	0	0	0	0	0	0	0
nreverse	0	0	0	0	0	0	0	0	0	0.214285714	0.214285714

SYSTEM COUPLING

	≀urn_simple_t	turn_type_na	turn_type_nu	single_fun_id	var_exist	rite_last_call_	copy_node	getdecls	nreverse
add_call_to_list	0	0	0	0	0	0	0	0	0
add_complex_type	0	0.072747776	0.014457831	0	0	0	0.027618776	0	0
add_field_to_list	0	0.059243872	0.028915663	0	0	0	0.013793103	0	0
add_para_to_list	0	0	0	0	0	0	0	0	0
add_simple_type	0.009803922	0.088254205	0.014883062	0	0	0	0.009892463	0	0
add_str_to_exp_va	0	0	0	0	0	0	0	0	0
add_type	0	0.064285714	0	0	0	0	0.041233766	0	0
add_var_to_list	0	0	0	0	0.032462866	0.001251565	0	0	0
brace_name	0	0	0	0	0	0	0	0	0
break_comp	0	0	0	0	0	0	0	0	0
dump_data	0	0.010658017	0.007993513	0	0.041331803	0.005938914	0	0.012019231	0.002403846
extend_var_list	0	0	0	0	0.013268465	0	0	0	0
filter_comp	0	0	0	0	0	0	0	0	0
get_code_name	0.166666667	0.166666667	0	0	0	0	0	0	0
et_component_nam	0	0	0	0	0	0	0	0	0
get_exp_var1	0	0	0	0	0	0	0	0	0
get_exp_var_list	0	0	0	0	0.050244134	0	0	0	0
get_last_call_id	0	0	0	0	0	0	0	0	0
get_system_name	0	0	0	0	0	0	0	0	0
get_type_id	0	0	0	0	0	0.007352941	0	0	0
nitialize_data_type	0	0.048192771	0.036144578	0	0	0	0	0	0
is_complex	0	0	0	0	0	0	0	0	0
is_expression	0	0	0	0	0	0	0	0	0
is_single_operand	0	0	0	0	0	0	0	0	0
make_field	0	0.051724138	0	0	0	0	0.034482759	0	0
make_fun_call	0	0	0	0	0	0	0	0	0
ake_parameter_no	0	0	0	0	0	0	0	0	0
make_var	0	0	0	0	0.025540041	0	0	0	0
mk_parm	0	0	0	0	0.084838963	0	0	0	0
modify_var	0	0	0	0	0.097427795	0	0	0	0
print_complex_typ	0	0	0	0	0	0	0	0	0
print_fun_calls	0	0	0	0	0	0	0	0	0
print_parameter_lis	0	0	0	0	0	0	0	0	0
print_simple_type	0	0.048192771	0.036144578	0	0	0	0	0	0
print_var_list	0	0	0	0	0	0	0	0	0
rocess_array_point	0.177777778	0.101094691	0	0	0	0	0.022374816	0	0
process_calls_parm	0	0	0	0	0	0	0	0	0
process_cond_exp	0	0	0	0	0.022556391	0	0	0	0
process_enum	0	0	0	0	0	0	0	0	0
ocess_one_call_pa	0	0	0	0	0	0	0.075	0	0
ocess_read_write_	0	0	0	0	0.0237691	0	0	0	0
rocess_single_par	0	0	0	0	0.02174701	0	0.002247191	0	0
quote_str	0	0.005783133	0.004337349	0	0.005373134	0.011764706	0	0	0.004285714
return_name	0	0	0	0	0	0	0	0	0.007936508
return_simple_typ	0.333333334	0.083333333	0	0	0	0	0	0	0
return_type_name	0.021739131	0.107895708	0.012572027	0	0	0	0.015206682	0	0
return_type_num	0	0.048192771	0.036144578	0	0	0	0	0	0
single_fun_id	0	0	0	0	0	0	0	0	0
var_exist	0	0	0	0	0.109720002	0	0	0	0
write_last_call_id	0	0	0	0	0	0.029411765	0	0	0
copy_node	0	0.058292283	0	0	0	0	0.089270372	0	0
getdecls	0	0	0	0	0	0	0	0.5	0
nreverse	0	0	0	0	0	0	0	0	0.071428571

SYSTEM COUPLING

References

[Aike94] Peter Aiken, Alice Muntz, and Russ Richards, ***DOD Legacy Systems Reverse Engineering Requirements***, Communication of the ACM No.5 Vol. 37 May 1994.

[Algh94] J.S. Alghamdi , **Programming Languages: A Quantitative Methodology for Assessments Software Metrics to Measure Syntactic Properties,** Ph. D. Thesis, Arizona State University , Aug. 1994.

[Alna97] S. Al-Nasser, **Measuring Coupling of Computer Programs**. Msc Thesis, KFUPM , Jun. 1997.

[Bach94] Richard Bache & Gualtiero Bazzana., **Software Metrics for Product Assessment.**, McGraw-Hill ,1994.

[Bigg94] Ted J. Biggerstaff, Bharat G. Mitbancer and Dallas E,. Webster, ***Program Understanding and the Concept Assignment Problem***. Communication of the ACM No.5 Vol. 37 May 1994.

[Boch91] Grady Booch, ***Object Oriented Design with Application,***. The Benjamin/Cummings Publishing Company, Inc. 1991.

[Brok87] Brooks, F. P, No Silver Bullet: Essence and Accidents of Software Engineering,. Computer, April 1987.

[Can90] Fazli Can and Esen A. Ozkarahan., **Concept and Effectiveness of the Cover-Coefficient-Based Clustering Methodology for Text Databases**. ACM Transactions on Database Systems Vol. 15, No.4 , Dec. 1990.

[Chen93] J. Chen, and J. Lu, A New Metric for Object Oriented Design,. Information and Software Technology, Vol. 35, no. 4, pp232-240, Apr 1993.

[Choi90] Song C. Choi and Walt Scachhi, ***Extracting and Restructuring the Design of Large Systems,***. IEEE Software January 1990.

[Cross09] Elliot J. Chikofsky , James H.Cross II , ***Reverse-Engineering and Design Recovery : A Taxonomy,***. IEEE Software January 1990.

[Dham95] H. Dhama, **Quantitative Models of Cohesion and Coupling in Software,** Journal of System and Software, Vol.29, no.1,pp.65-74,Apr.1995.

[Fent90] N. Fenton and A. Melton, **Deriving Structurally Based Software Measures,** Journal of System and Software, Vol.12, no.1 pp177-187, Apr 1990.

[Fent91] Norman E. Fenton , **Software Metric: A Rigorous Approach**, Chapman & Hall., 1991

[Forte92] Gene Forte, Tools Fair : Out of the Lab onto the Shelf. IEEE Software May 1992.

[Gill92] Allan Gillies, **Software Quality, Theory and Management.**, Chapman & Hall Computing, 1992.

[Good94] Paul Goodman, **Practical Implementation of Software Metrics,**. McGraw-Hill ,1994.

[Hall92] P.A.V. Hall, Software Reuse and Reverse Engineering in Practice,. Chapman & Hall 1992.

[Hara90] Mehdi T. Harandi and Jim Q. Ning , Knowledger-Based Program Analysis. IEEE Software January 1990.

[Haus90] Philip A, Hausler and Mark G. Pleszkoch, Richard C. Linger and Alan R, Hevner, ***Using Function Abstraction to Understand Program Behavior***,. IEEE Software 1990.

[Horw90] Susan Horwitz, Thomas Reps, and David Binkely, ***Interprocedureal Slicing Using Dependence Graphs***. ACM Transactions of Programming Languages and Systems, Vol. 12, No. 1, Jan 1990. pp. 26-60.

[Hutt94] Andrew T.F. Hutt, Object analysis and design descriptions of Methods. Object Management Group 1994.

[lieb93] Karl J. Lieberherr, Ignacio Silva-Lepe, Cun Xiao, ***Adaptive Object-Oriented Programming using Graph-Based Customization,***. Communication of the ACM No.5 Vol. 37 May 1994.

[Mark94] Lawrence Markosiam, Philip Newcomb, Russel Brand, Scott Burson, and Ted Kitzmiller, ***Using an Enabling Technology to Reengineer Legacy Systems,***. Communication of the ACM No.5 Vol. 37 May 1994.

[Maye[93] Herbert G. Mayer and Michael Wolfe, ***InterProcedural Alias Analysis: Implementation and Empirical Results***. Software Practice and Experience vol. 23(11). 1201-1233(Nov. 1993).

[Moll93] K.H. Moller and D.J Paulish., **Software Metrics**, Chapman & Hall., 1993

[Ning92] Jim Q. Ning, Andre Engberts, and W Kozaczynski, Automated Support for legacy code understanding.

[Offu93] A.J. Offut, M.J. Harrold and P. Kotle, **A Software Metric System for Module Coupling,** Journal of System and Software, Vol. 20, no.3, pp 295-308 Mar 1993.

[Pame90] Pamela Samuelson, Reverse-Engineering Someone Else's software: Is It Legal?,. IEEE Software January 1990.

[Pitt93] Matthew Pittman, Lessons Learned in Managing Object-Oriented Development. IEEE Software January 1993.

[Pitt93] Pittman,M., Lessons Learned in Managing Object-Oriented Development,. IEEE Software , January 1993.

[Prem94] William J. Premerlani and Michael R. Blaha. ***An approach for Reverse engineering of relational database,.*** Communication of the ACM No. Vol. 37 May 1994.

[Quil94] Alex Quilici, ***A Memory-Based Approach to Recognizing Programming Plans***. Communication of the ACM No.5 Vol. 37 May 1994.

[Rich90] Charles Rich and Linda Wills, Recognizing a Program's Design: A Graph-Parsing Approach,. IEEE Software January 1990.

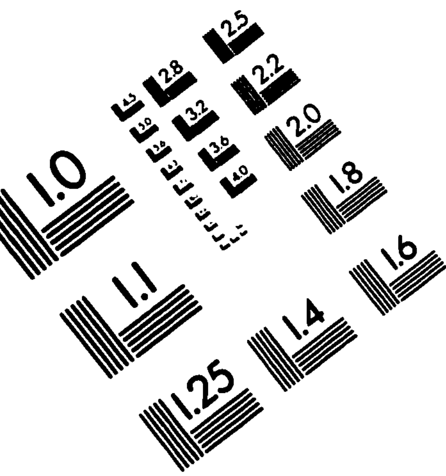
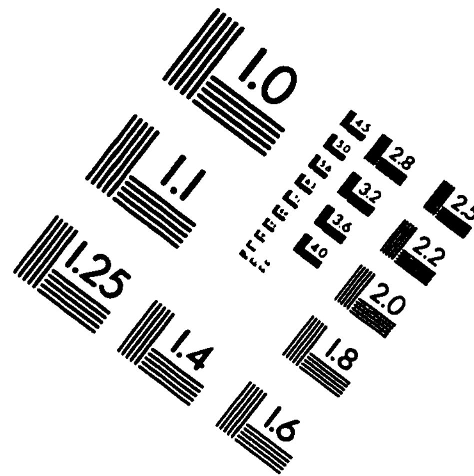
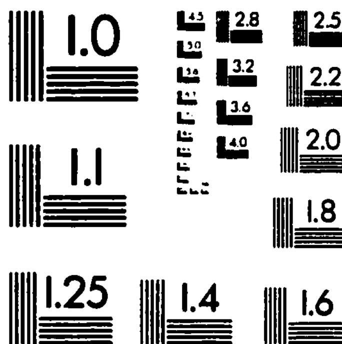
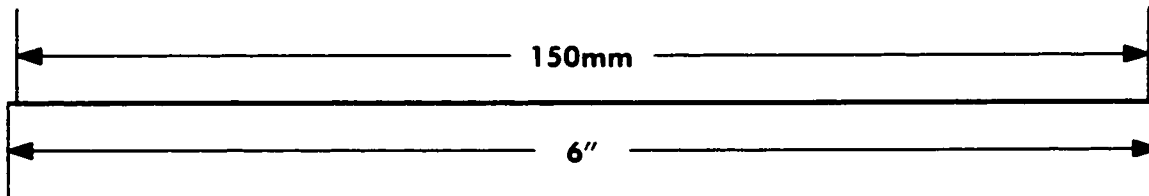
[Robi92] Peter Robinson, ***Object-Oriented Design***,. Chapman & Hall, 1992.

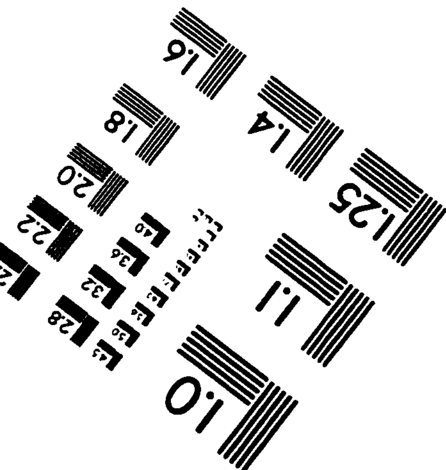
[Ruga90] Spencer Rugaber, Stephen B. Ornburn and Richard J. LeBlanc, IEEE Software January 1990.

[Sned95] Harry M. Sneed,. **Understanding Software Through Numbers: A Metric Based Approach to Program Comprehension**. Software Maintenance :Research and practice vol. 7. (1995).

[Voas93] Jeffrey M. Voas and Keith w. Miller. **Semantic Metrics for Software Testability**, .Journal of system Software , vol 20, p 207-215, 1993.

IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved