

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

1.

(a)

I give permission for my thesis, entitled

COBOL Program Analyser and
Workbench - - - - Phase 1

to be made available to readers in the Library under the conditions determined by the Librarian.

(b)

I agree to my thesis, if asked for by another institution, being sent away on temporary loan under conditions determined by the Librarian.

(c)

I also agree that my thesis may be copied for Library use.

2.

I do not wish my thesis, entitled

~~COBOL Program Analyser and
Workbench - - - - Phase 1~~

to be made available to readers or to be sent to other institutions without my written consent within the next two years.

Signed *[Signature]*
Date 12/9/85 *[Signature]*

Strike out the sentence or phrase which does not apply.

The Library
Massey University,
Palmerston North, N.Z.

The copyright of this thesis belongs to the author. Readers must sign their name in the space below to show that they recognise this. They are asked to add their permanent address.

Name and Address	Date
.....
.....
.....
.....
.....



**COBOL PROGRAM ANALYSER
AND
WORKBENCH --- PHASE I**

A thesis presented in partial fulfilment
of the requirements for
the degree of **MASTER of SCIENCE**
in **COMPUTER SCIENCE**
at **MASSEY UNIVERSITY**

by

TAN, KAY MENG B Sc (Hons)

August 1985.

ABSTRACT

This dissertation addresses a number of important software maintenance problems which fall within the framework of software management and software engineering. Compiling techniques were used to present in a standard form the structure of the COBOL program PROCEDURE DIVISION; as well as providing a framework for some complexity measures of a given COBOL program. The latter can give some measure of the ease of software modification during the software maintenance phase.

The purpose of the project of which this thesis is a part is to build up a database which will completely specify a COBOL program and enable it to be analysed in various ways related to maintenance, enhancement and other tasks.

A COBOL program information system (MJCAS) has been designed for use by the applications programmer, to acquire the information about a COBOL program (e.g. complexity measures, control flow analysis) by analysing the structure of the COBOL program. Various software methods such as structured programming, structured design, top-down design and bottom-up program testing were used throughout the MJCAS system development process. The finite-state machine concept was used to construct a **COBOL lexical scanner**. The COBOL programming syntax was parsed by the top-down recursive-descent method. The COBOL program was reformatted according to a coding standard to show the structure of the COBOL program.

Many complexity measures are mentioned in this thesis. A combination complexity measure method and guidelines are suggested. A COBOL program complexity factor is proposed. In general, the program complexity measure could apply to other programming languages.

ACKNOWLEDGEMENTS

Firstly, I thank my supervisor **Prof. Graham Tate** for his constructive efforts and supervision.

Secondly, I am grateful to those people who also assisted and directed my research. Although there are many who give me both ideas and encouragement, I specifically thank **Mr. Peter Parkin**, Mr. J. L. Alexander, Mrs. Nola M. Simpson, Mr. Chris Philips and Miss June M. Verner.

Thirdly, I acknowledge the many Systems Analysts, Managers, Computer Scientists and Computer programmers/operators who have enriched me with many ideas of relevance to this project during my association with them.

I appreciate the assistance by Mr. Kevin Kelliher and Mr. Brain Kirk for their proof-reading of my thesis.

I also thank my family for their financial and moral support which made this opportunity for further study possible.

Finally, I wish to recognise the following friends who helped me in the preparation of this thesis:

Mr. Ooi, Inn Tong

Mr. Du, Yuchun

Mrs. Ooi, Phaik Kim

Mr. Walt Abell

Miss Phuah, Phaik Lean

Mr. Giovanni Moretti

Mr. Wu, Chiaw Ching

Mr. John Holley

LIST OF FIGURES

<u>Figure</u>	<u>Title</u>	<u>Page</u>
P.1	What This Thesis is About.	P - 1
1.1	The COBOL Program Processing Activities in a MUCAS Aided Production Project.	3
3.1	Datapro Survey : Percentage Usage of Languages in 5813 Data Processing Centers [GEP77].	20
4.1	MUCAS Implementation Procedure.	24
4.2	MUCAS System Overview Structure Diagram.	33
4.3	MUCAS HOUSEKEEPING Module Structure Diagram.	33
4.4	MUCAS FIRST PASS Module Structure Diagram.	34
4.5	MUCAS SECOND PASS Module Structure Diagram.	35
4.6	MUCAS TERMINATION Module Structure Diagram.	35
4.7	Functional Modules of MUCAS.	38
4.8	Major Parts Of COBOL Programs.	54
4.9	COBOL Program Structure.	56
4.10	COBOL Program Elements Hierarchy.	56
4.11	COBOL Statement Type Complexity Measure.	66
4.12	A paragraph of a simple program complexity measure report.	67
4.13	A simple program complexity measure report.	67
C.1	COBOL Source Program Tram-Line Diagrams.	C - 1
C.2	COBOL Source Program Tram-Line Diagrams (Continued).	C - 2
C.3	COBOL Source Program Tram-Line Diagrams (Continued).	C - 3
C.4	COBOL Statement Type Tram-Line Diagrams.	C - 4
C.5	COBOL Statement Type Tram-Line Diagrams (Continued).	C - 5
C.6	COBOL Statement Type Tram-Line Diagrams (Continued).	C - 6
C.7	COBOL Statement Tram-Line Diagrams.	C - 7
C.8	COBOL Statement Tram-Line Diagrams (Continued).	C - 8
C.9	COBOL Statement Tram-Line Diagrams (Continued).	C - 9
C.10	COBOL Statement Tram-Line Diagrams (Continued).	C - 10
C.11	COBOL Statement Tram-Line Diagrams (Continued).	C - 11
C.12	COBOL Statement Tram-Line Diagrams (Continued).	C - 12
C.13	COBOL Statement Tram-Line Diagrams (Continued).	C - 13
C.14	COBOL Statement Tram-Line Diagrams (Continued).	C - 14
C.15	COBOL Statement Tram-Line Diagrams (Continued).	C - 15
C.16	COBOL Statement Tram-Line Diagrams (Continued).	C - 16

<u>Figure</u>	<u>Title</u>	<u>Page</u>
C.17	COBOL Statement Tram-Line Diagrams (Continued).	C - 17
C.18	COBOL Statement Tram-Line Diagrams (Continued).	C - 18
C.19	COBOL Statement Tram-Line Diagrams (Continued).	C - 19
C.20	COBOL Statement Tram-Line Diagrams (Continued).	C - 20
C.21	COBOL Statement Tram-Line Diagrams (Continued).	C - 21
C.22	COBOL Statement Tram-Line Diagrams (Continued).	C - 22
C.23	COBOL Statement Tram-Line Diagrams (Continued).	C - 23
C.24	COBOL Statement Tram-Line Diagrams (Continued).	C - 24
C.25	COBOL Statement Tram-Line Diagrams (Continued).	C - 25
C.26	COBOL Statement Tram-Line Diagrams (Continued).	C - 26
C.27	COBOL Statement Tram-Line Diagrams (Continued).	C - 27
C.28	COBOL Statement Tram-Line Diagrams (Continued).	C - 28
C.29	COBOL Statement Tram-Line Diagrams (Continued).	C - 29
C.30	COBOL Statement Tram-Line Diagrams (Continued).	C - 30
C.31	COBOL Statement Tram-Line Diagrams (Continued).	C - 31
C.32	COBOL Statement Tram-Line Diagrams (Continued).	C - 32
C.33	COBOL Statement Tram-Line Diagrams (Continued).	C - 33
C.34	COBOL Statement Tram-Line Diagrams (Continued).	C - 34
C.35	COBOL Statement Tram-Line Diagrams (Continued).	C - 35
C.36	COBOL Statement Tram-Line Diagrams (Continued).	C - 36
C.37	COBOL Statement Tram-Line Diagrams (Continued).	C - 37
C.38	COBOL Condition Tram-Line Diagrams.	C - 38
C.39	COBOL Condition Tram-Line Diagrams (Continued).	C - 39
C.40	COBOL Condition Tram-Line Diagrams (Continued).	C - 40
C.41	COBOL Condition Tram-Line Diagrams (Continued).	C - 41
C.42	COBOL Qualification Tram-Line Diagrams.	C - 42
C.43	COBOL Qualification, Subscripting and Reference-modification Tram-Line Diagrams.	C - 43
C.44	COBOL Identifier Tram-Line Diagrams.	C - 44
D.1	MUCAS System Overview Structure Diagram.	D - 1
D.2	MUCAS HOUSEKEEPING Module Structure Diagram.	D - 1
D.3	MUCAS FIRST PASS Module Structure Diagram.	D - 2
D.4	MUCAS SECOND PASS Module Structure Diagram.	D - 3
D.5	MUCAS TERMINATION Module Structure Diagram.	D - 4
D.6	Scan NEXT SYMBOL Structure Diagram.	D - 4

<u>Figure</u>	<u>Title</u>	<u>Page</u>
D.7	SCAN SYMBOL Structure Diagram.	D - 5
D.8	Parsing COBOL PROCEDURE DIVISION Structure Diagram.	D - 6
D.9	Structure Diagram for Parsing a COBOL Section.	D - 7
D.10	Structure Diagram for Parsing a COBOL Paragraph.	D - 8
D.11	Structure Diagram for Parsing a COBOL USE-Section.	D - 9
D.12	Structure Diagram for Parsing a COBOL Statement.	D - 10
D.13	COBOL Nucleus Statement Structure Diagram.	D - 11
D.14	Sequential I/O Statement Structure Diagram.	D - 12
D.15	Relative I/O Statement Structure Diagram.	D - 12
D.16	Index I/O Statement Structure Diagram.	D - 12
D.17	Inter-Program Statement Structure Diagram.	D - 13
D.18	Sort/Merge Statement Structure Diagram.	D - 13
D.19	Debug Statement Structure Diagram.	D - 13
D.20	Report Writer Statement Structure Diagram.	D - 14
D.21	Communication Statement Structure Diagram.	D - 14
D.22	Assignment Statement Structure Diagram.	D - 14
D.23	Condition Statement Structure Diagram.	D - 15
D.24	Compound Statement Structure Diagram.	D - 15
E.1	The Test Input COBOL Source Program.	E - 1
E.2	The Test Input COBOL Source Program (Continued).	E - 2
E.3	The Reformatted Sample COBOL Source Program.	E - 3
E.4	The Reformatted Sample COBOL Source Program (Continued).	E - 4
E.5	The Reformatted COBOL Source Program Paragraph 0010-MAINLINE Programming Syntax Complexity Measure Report.	E - 5
E.6	The Reformatted COBOL Source Program Paragraph 0100-INITIALIZATION Programming Syntax Complexity Measure Report.	E - 5
E.7	The Reformatted COBOL Source Program Paragraph 0200-MAIN-PROCESSING Programming Syntax Complexity Measure Report.	E - 6
E.8	The Reformatted COBOL Source Program Paragraph 0900-TERMINATION Programming Syntax Complexity Measure Report.	E - 6
E.9	The Reformatted COBOL Source Program Paragraph 1000-LOOP-BEGIN Programming Syntax Complexity Measure Report.	E - 7
E.10	The Reformatted COBOL Source Program Paragraph 2000-CHECK-PROCESS-READ Programming Syntax Complexity Measure Report.	E - 7

<u>Figure</u>	<u>Title</u>	<u>Page</u>
E.11	The Reformatted COBOL Source Program Paragraph 3000-LOOP-END Programming Syntax Complexity Measure Report.	E - 8
E.12	The Reformatted COBOL Source Program Paragraph 4000-CHECK-DATE Programming Syntax Complexity Measure Report.	E - 8
E.13	The Reformatted COBOL Source Program Paragraph 4100-PROCESS-DATA Programming Syntax Complexity Measure Report.	E - 9
E.14	The Reformatted COBOL Source Program Paragraph 4200-READ-DATA-TEST Programming Syntax Complexity Measure Report.	E - 9
E.15	The Reformatted COBOL Source Program Paragraph 4300-WRITE-LINE Programming Syntax Complexity Measure Report.	E - 10
E.16	The Reformatted COBOL Source Program Paragraph 4400-WRITE-ERROR Programming Syntax Complexity Measure Report.	E - 10
E.17	The Reformatted COBOL Source Program Paragraph 4500-WRITE-SUMMARY Programming Syntax Complexity Measure Report.	E - 11
E.18	The Reformatted COBOL Source Program Programming Syntax Complexity Measure Report.	E - 12

ABBREVIATIONS

ACM	Association for Computing Machinery
ADP	Automatic Data Processing
AFIPS	American Federation of Information Processing Societies
ANS	American National Standards
ANSI	American National Standards Institute
BCS	British Computer Society
BNF	Backus-Naur Form
CAD	Computer-Aided Design
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacturing
CAI	Computer-Aided Instruction
CAI	Computer-Assisted Instruction
CICS	Customer Information Control System
CF	Control Flow
CFG	Control Flow Graph
CLS	Cobol Lexical Scanner
CM	Complexity Measure
COBOL	Common Business Oriented Language
CODASYL	Conference on Data Systems Languages
CPU	Central Processing Unit
CVC	Control Variable Complexity
DBA	Direct Bucket Address
DBA	DataBase Administrator
DBMS	DataBase Management System
DMS	Data Manipulation System
EDP	Electronic Data Processing
FIPS	Federation Information Processing Standards
IBM FSD	IBM Federal Systems Division
HGL	Hierarchical Graph Language
HIPO	Hierarchy plus Input-Process-Output
IBM	International Business Machines
ICL	International Computers Ltd.
IDS	Integrated Data Store
IEEE	Institute of Electrical and Electronic Engineers
IFIP	International Federation for Information Processing
LL	Leftmost Looking grammar
LSI	Large-Scale Integration
MC	Module Complexity
MUCAS	Massey University Cobol program Analyser System
PC	Program Complexity
PCM	Program Complexity Measure
PL/I	Programming Language I
PSL/PSA	Problem Statement Language/Problem Specification Analyser
SD	Structured Design
SIGPLAN	Special Interest Group on Programming Languages (of ACM)
SP	Structured Programming
SRE	Software Requirements Engineering
ST	Symbol Table
UNIVAC	UNIVERSAL Automatic Computer
VLSI	Very Large-Scale Integration

TABLE OF CONTENTS	PAGE
TITLE PAGE	
ABSTRACT.....	i
ACKNOWLEDGEMENTS.....	ii
LIST OF FIGURES.....	iii TO vi
ABBREVIATIONS.....	vii
TABLE OF CONTENTS.....	viii TO xi
ABOUT THE THESIS.....	xii
CHAPTER 1 INTRODUCTION.....	1
CHAPTER 2 COMPUTER PROGRAM FLOW ANALYSIS AND APPLICATIONS.....	7
2.1 DESCRIPTION OF FLOW ANALYSIS.....	7
2.2 DISCUSSION OF CONTROL FLOW ANALYSIS.....	8
2.3 DIFFICULTIES AND POSSIBLE SOLUTIONS IN BUSINESS DATA PROCESSING.....	9
2.4 PROGRAM COMPLEXITY MEASURES.....	12
2.5 A TOOL FOR OBTAINING SOME COBOL PROGRAM COMPLEXITY MEASURE (MUCAS).....	13
2.5.1 BASIC CONTROL FLOW STRUCTURE FOR COBOL.....	14
2.5.2 MUCAS DESIGN TECHNIQUES.....	15
2.5.3 MUCAS FORMATTING STANDARDS.....	16
2.6 APPLICATION AREA OF MUCAS.....	17
2.6.1 REFORMATTING COBOL PROGRAMS.....	17
2.6.2 COBOL PROGRAM COMPLEXITY MEASURES.....	18
2.6.3 COBOL PROGRAM MAINTENANCE.....	18
CHAPTER 3 REVIEW OF THE LITERATURE.....	19
3.1 SURVEY OF THE SOFTWARE LIFE CYCLE.....	19
3.2 THE CURRENT USAGE OF PROGRAMMING LANGUAGES.....	20
3.3 TRENDS IN PROGRAMMING LANGUAGES.....	21
3.4 THE PROBLEMS OF SOFTWARE RELIABILITY.....	21
3.5 IMPLICATIONS FOR MASSEY UNIVERSITY COBOL ANALYSER SYSTEM.....	22
CHAPTER 4 THE DESIGN METHOD OF MUCAS.....	24
4.1 FUNCTIONAL DECOMPOSITION OF MUCAS.....	24
4.2 THE METHOD OF IMPLEMENTING MUCAS.....	25
4.3 TRENDS IN SYSTEMS DESIGN.....	27
4.3.1 THE PROGRAMMING PROCESS.....	27
4.3.2 FUNCTIONAL DEFINITION AND INTERFACES.....	27
4.3.3 TOP-DOWN DESIGN AND BOTTOM-UP TESTING.....	29
4.4 THE MUCAS DESIGN.....	30
4.4.1 AN OVERVIEW OF THE STRUCTURE OF MUCAS.....	31
4.4.2 THE HOUSEKEEPING FUNCTIONS OF MUCAS.....	36

4.4.3	THE VARIOUS MODULES OF MUCAS.....	37
4.4.4	THE PARSING FORMATTING AND COMPLEXITY MEASURE OF MUCAS.....	39
4.4.4.1	THE LEXICAL ANALYSIS STAGES OF MUCAS.....	39
4.4.4.2	SYMBOL TABLE CONSTRUCTION.....	43
4.4.4.2.1	SYMBOL TABLE LOOKUP TECHNIQUES.....	44
4.4.4.2.2	RELATE SYMBOL TO NAME LIST.....	46
4.4.4.3	THE SYNTAX ANALYSIS STAGES OF MUCAS.....	49
4.4.4.3.1	MUCAS PARSING METHOD.....	50
4.4.4.3.2	TOP-DOWN RECURSIVE-DESCENT METHOD.....	51
4.4.4.3.3	TOP-DOWN SYNTAX-DIRECTED METHOD.....	52
4.4.4.3.4	BOTTOM-UP METHOD.....	53
4.4.4.3.5	THE STRUCTURE OF THE PROCEDURE DIVISION OF COBOL.....	53
4.4.4.3.6	COBOL STATEMENT ANALYSIS.....	57
4.4.4.3.6.1	COBOL STATEMENTS.....	57
4.4.4.3.6.2	TYPES OF STATEMENTS IN COBOL.....	57
4.4.4.3.6.3	OBTAIN STATEMENT TYPE.....	58
4.4.4.3.6.4	PROCESS STATEMENT TYPE.....	58
4.4.4.4	FORMATTING ANALYSIS STAGES OF MUCAS.....	59
4.4.4.4.1	METHODOLOGY OF FORMATTING.....	59
4.4.4.4.2	LEVEL OF INDENTATION DESIGN.....	60
4.4.4.5	PROGRAM COMPLEXITY MEASURE ANALYSIS STAGES OF MUCAS.....	62
4.4.4.5.1	BACKGROUND AND METHODOLOGY.....	63
4.4.4.5.2	PROGRAMMING SYNTAX COMPLEXITY CONTRIBUTION TO PC.....	64
4.5	DISCUSSION ON MUCAS WRITING.....	68
4.5.1	MUCAS WRITING OVERVIEW.....	68
4.5.2	INTERESTING FEATURES OF MUCAS WRITING.....	69
4.5.3	THE RELIABILITY AND EFFICIENCY OF MUCAS.....	70
4.5.4	THE FLEXIBILITY OF MUCAS.....	70
4.6	DIAGNOSTIC FACILITIES AND IMPLEMENTATION TECHNIQUE.....	71
4.6.1	ERROR DIAGNOSTICS.....	73
4.6.2	COMPILATION TIME ERROR MESSAGES.....	73
4.6.3	TRACING AND SYMBOL TABLE DUMP.....	74
4.6.4	SKIP TO SPECIFIED SOURCE.....	74
4.7	POSSIBLE MODIFICATIONS OF MUCAS.....	74
CHAPTER 5	DESIGN PROBLEMS, GUIDELINES AND POSSIBLE MUCAS DEVELOPMENTS.....	76
5.1	OUTSTANDING DESIGN PROBLEMS.....	76
5.1.1	FACTORS OF PROGRAMMER DEMOTIVATION IN COBOL PROGRAM MAINTENANCE.....	77
5.2	COBOL ENVIRONMENT SOFTWARE QUALITY GOALS.....	78
5.3	INCORPORATE GRAPHIC FACILITY INTO MUCAS SYSTEM.....	80
5.4	IMPROVEMENT OF MUCAS SYSTEM OBJECTIVES.....	81
5.4.1	PROVING COBOL PROGRAM CORRECTNESS.....	82

5.4.2	COMPLEXITY MEASURE ANALYSIS.....	85
5.4.2.1	COBOL PROGRAM STRUCTURAL ISSUES.....	87
5.4.2.2	A METHOD TO DEFINE A WELL-STRUCTURED PROGRAM.....	90
5.4.3	COBOL PROGRAM COMPLEXITY MEASURE.....	92
5.4.3.1	CONTROL VARIABLE COMPLEXITY MEASURE.....	94
5.4.3.2	MODULE COMPLEXITY MEASURE.....	94
5.5	SOME COBOL PROGRAM MAINTENANCE PROBLEMS AND CODING GUIDELINES.....	97
5.5.1	CODING GUIDELINES FOR COBOL PROGRAM.....	98
5.6	OTHER INFLUENCES ON THE MUCAS DESIGN.....	100
CHAPTER 6	DISCUSSION AND CONCLUSION.....	102
6.1	CAUSES OF PROBLEMS IN COMPUTER PROGRAMMING.....	102
6.1.1	THE CHANGES IN INPUT MEDIUM AND COMPUTER PROGRAMMING.....	102
6.1.2	COMPUTER PROGRAMMING DIFFICULTIES.....	103
6.1.3	EXPERIENCE FROM THE REVISION OF ANS COBOL-80.....	104
6.1.4	THE PROGRAM DESIGN INCOMPETENCE PROBLEM.....	105
6.2	SOFTWARE MAINTENANCE PROBLEMS AND THE EFFECT OF MUCAS TYPE SOFTWARE APPLICATIONS.....	107
6.3	THE SUMMARY AND CONCLUSIONS.....	110
APPENDIX A	: COBOL RESERVED WORDS AND SYMBOLS.....	A-1 TO A-4
APPENDIX B	: COBOL PROGRAMMING LANGUAGE SYNTAX.....	B-1
I.	NOTATIONAL CONVENTIONS.....	B-1
II.	LANGUAGE CONCEPTS.....	B-2
(A)	Character Set.....	B-2
(B)	Separators.....	B-2
(C)	Character Strings.....	B-2
(D)	Uniqueness of Reference.....	B-3
(E)	Reference Format.....	B-3
(F)	Source Program Structure.....	B-3
(G)	IDENTIFICATION DIVISION.....	B-3
(H)	ENVIRONMENT DIVISION.....	B-4
(I)	DATA DIVISION.....	B-4
(J)	PROCEDURE DIVISION.....	B-5
III.	THE COBOL GRAMMAR.....	B-5
IV.	LOW LEVEL SYNTAX OF COBOL.....	B-6
V.	GENERAL FORMAT FOR COBOL SOURCE PROGRAM.....	B-7
VI.	GENERAL FORMAT FOR PROCEDURE DIVISION.....	B-8
VII.	GENERAL FORMAT FOR COBOL VERBS.....	B-10
VIII.	GENERAL FORMAT FOR CONDITIONS.....	B-26
1.	RELATION CONDITION.....	B-26
2.	CLASS CONDITIONS.....	B-26
3.	CONDITION-NAME CONDITIONS.....	B-26
4.	SWITCH-STATUS CONDITIONS.....	B-26

5. SIGN CONDITIONS.....	B-26
6. NEGATED CONDITIONS.....	B-26
7. COMBINED CONDITIONS.....	B-26
8. ABBREVIATED COMBINED RELATION CONDITIONS.....	B-26
IX. GENERAL FORMAT FOR QUALIFICATION.....	B-27
X. MISCELLANEOUS FORMATS.....	B-27
1. SUBSCRIPTING :.....	B-27
2. REFERENCE MODIFICATION :.....	B-27
3. IDENTIFIER :.....	B-27
APPENDIX C : COBOL SYNTAX TRAM-LINE DIAGRAM.....	C-1 TO C-44
APPENDIX D : MUCAS STRUCTURE DIAGRAM.....	D-1 TO D-15
APPENDIX E : MUCAS RUN EXAMPLE.....	E-1 TO E-12
APPENDIX F : BIBLIOGRAPHY AND REFERENCES.....	F-1 TO F-43
APPENDIX G : MANUALS.....	G-1 TO G-3
APPENDIX H : INDEX	H-1 TO H-11

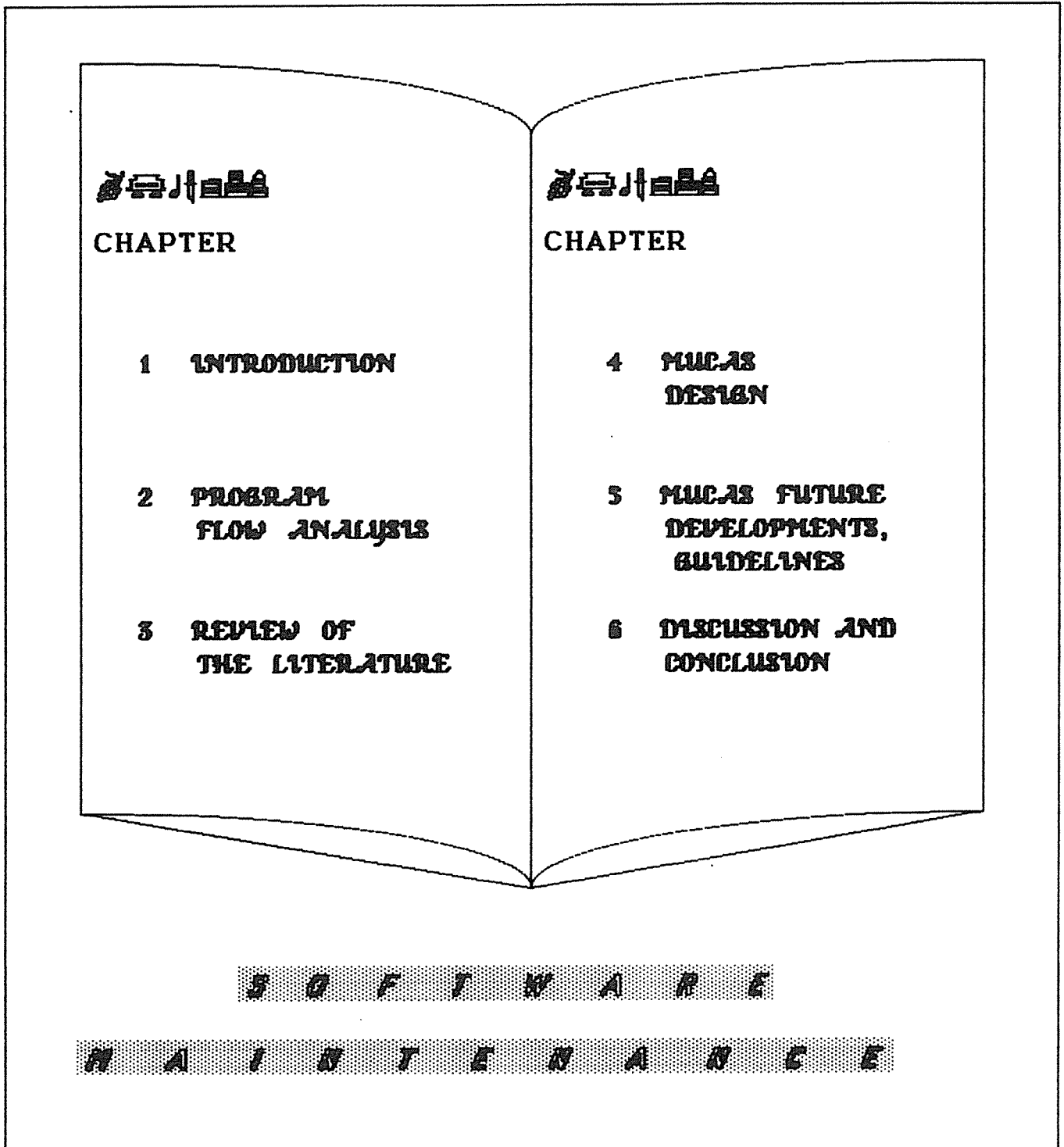


Figure P.1 What This Thesis is About.

CHAPTER 1

INTRODUCTION

CHAPTER 1 INTRODUCTION

The areas of **Software Management** and **Software Engineering** aim for an understanding of how to achieve reliability in software maintenance [BOE79] [FIS80]. There has been a worldwide software maintenance problem revealed by the computing literature ([WEX84] and Chapter 3 of this thesis). Too many production systems have been developed which are very costly or impossible to maintain [POD85] [MOR85]. Too often systems have become unmaintainable making it necessary to write new systems and to start another systems development cycle [GLA81].

For almost 30 years of computing history, the hardware cost has decreased greatly, whereas in recent years, the software cost has increased steadily [BRO82] [FOX82]. At present the majority of the software cost (about 70%) is due to maintenance [LEA83]. There is a need for new software tools which will help to solve software maintenance problems. The most significant problem of software maintenance is verifying the correctness of software modifications [BOY81] :

- (a) What techniques can be used to implement this ?
- (b) Are these techniques feasible to apply ?
- (c) What are the implementation considerations ?

One attempt to solve these problems is the development of the ad hoc retest technique which uses graph theory [BEI78] and **0-1 integer** programming [FIS80], to examine performance characteristics, and practical implementation considerations. Although various methods have been tried, no universally acceptable approach for software modification and software maintenance has been agreed upon. A COBOL program information system **MUCAS** (**M**assey **U**niversity **C**obol program **A**nalyser **S**ystem) has been designed for use by the applications programmer, to acquire information about a COBOL program (e.g., the

complexity measure, control flow analysis) by analysing and reformatting the COBOL program. Developments of MUCAS will both advance computer software theory and benefit the management of the software maintenance process. The current computer complexity measure method approach to computer program testing will be extended to computer program maintenance. Practical benefits are anticipated to be significant because now managers will have a tool to analyse COBOL program. MUCAS will increase confidence that modifications to one module of a COBOL program do not affect the proper execution of the entire software system.

This dissertation addresses a number of important software maintenance problems which fall within the framework of software management and software engineering. Compiling techniques are used to analyse the COBOL program, from a view point of both reformatting the COBOL program PROCEDURE DIVISION and deriving a complexity measure of a given COBOL program, to assist in assessing the ease of software modification made during the software maintenance phase.

The purpose of the project of which this thesis is a part of build up a database which will completely specify a COBOL program and allow it to be analysed in various ways related to maintenance, enhancement and other tasks.

The used of MUCAS in a COBOL production project could also significantly ease the COBOL software maintenance process. Figure 1.1 shows the position of MUCAS in a modern COBOL production environment. Upon receiving the system change specification (come originally from user) from the project personnel (may be manager, project leader or systems analyst), the systems analyst or programmer will analyse the existing applications programs and design the program change specification. MUCAS will be used to analyse the existing programs which are to be modified in order to help determine both their existing

structure (and its quality) and also the extent of the structural changes necessary to implement the required modifications. This is followed by the programmer code creation according to the program change specification. While testing and verify output results using the test data and live data, any error will be feedback to redesign, retest or re-confirm user request accordingly until the program changes are correct and the modified program performs it specific task [BEI83]. At this stage, MUCAS will be used again to analyse it, generate the program analysis report and store the program information in the program database. If the program analysis report was satisfactory, the program changes are considered as successfully done; otherwise, the re-analysis of the existing program procedure will be repeated.

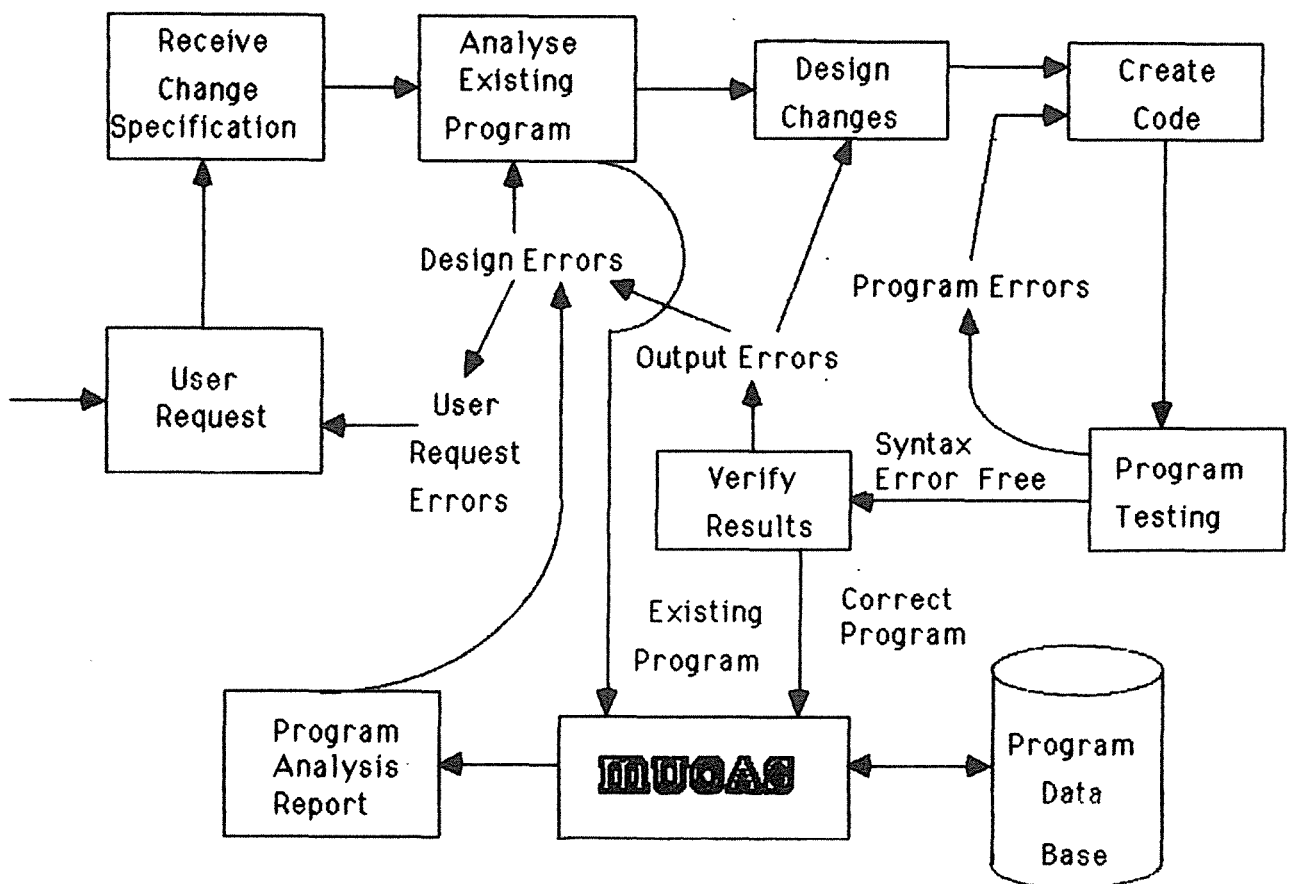


Figure 1.1 The COBOL Program Processing Activities
In a MUCAS Aided Production Project.

The writing of MUCAS mainly involved the computing techniques : compiler design techniques ([WEI73], [TRA63], [SEV74], [PRI71], [MOR68], [LUM71], [GLA69], [SIM73a], [COB68], [COB74], [COB81], [COR81], [HOP69], [MCK74], [AHO77], [GRI71] and [GRO74]), structured programming ([DAH72], [DOR72a], [DOR72b], [BAK72], [MCC78], [BAK74], [DIJ76], [ELS76], [LIN79] and [TRI79]), structured design techniques ([STE74], [WIR74], [RIC80], [OGD72], [NIC75], [MYE76] and [MYE78]), data base techniques ([OLL78], [OKO85], [CAR79a], [CHA77], [DAT81], [FER83] and [VET81]), software systems design ([BRO82], [AMS76], [ART84], [HAL77], [KER79], [KRU84], [LAN79], [LEA83], [MET81], [CAR81], [MCG80], [BAB85] and [JAC70]), program computational complexity ([OTH81], [SCH81], [WOO79], [ZOL80], [MCC76a], [BAK80], [CHA79], [CHE78], [GON84], [BAC57] and [MCT80]) and computer program testing ([BAS80], [BEI83], [BOY81], [GLA81], [GOU81] and [MUC81]).

Development of Structured Programming (SP) has evolved in conjunction with the top-down concept [MCC78]. MUCAS is designed to analyse the composition of COBOL programs so that it can not only show the structure of unstructured COBOL programs but can also test how good the structure is. **Testing** of MUCAS is bottom-up while **design** is top-down stepwise refinement [MCC78].

Each procedure was tested individually until it was error free and performed the correct functions, then it was added to the MUCAS system. Various software methods such as SP, structured design, top-down stepwise refinement design [MCC78] and bottom-up program testing were used throughout the MUCAS system development process [BEL76] [BAS75]. The finite-state machine concept [AHO77] was used to construct a COBOL lexical scanner. The COBOL programming syntax was parsed by the top-down recursive-descent method ([GRI71], [MCG80], [MCK74] and [AHO77]). The COBOL program was reformatted according to a coding standard to show the structure levels of the COBOL program.

Generally, the purpose of the MASSEY UNIVERSITY COBOL PROGRAM ANALYSER SYSTEM (MUCAS) AND WORKBENCH PHASE I is

- (i) To process the text of an existing COBOL program (which has previously been compiled and run, and therefore has no syntax errors) and identify all major structural components and relationships.
- (ii) To format the program in a standard manner (A coding standard is suggested in this thesis).
- (iii) To output tables and files of structural information for later use in the construction of the program specification database.

Many complexity measure methods are mentioned in this thesis. A combination complexity measure method and guidelines are suggested. A COBOL programming syntax complexity factor is proposed. In general, this type of program complexity measure could be applied to other programming languages.

The design goals of the MUCAS were :

(1) **Research**

To investigate the extent to which structural information relevant to the maintenance function can be derived automatically from existing COBOL programs.

(2) **Development**

To enable application maintenance programmers to check program logic and complexity of a given program.

(3) **Economics**

To reduce software maintenance effort and cost.

This thesis consists of six chapters. The first chapter is the introduction. Chapter 2 summarises the relevant concepts and methods. Chapter 3 reviews the literature and describes some of the history and current status of software trends in business data processing. Chapter 4 deals with design rules and algorithms for reformatting COBOL's PROCEDURE DIVISION and covers the implementation of MUCAS. Chapter 5 discusses future developments of MUCAS Phase I. Chapter 6 is the discussion and conclusion.