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 ...... Kay Meng, Tan.

Date 12/9/85

Strike out the sentence or phrase which does not apply.

Name and Address Date

....

....

....

....

....

FLOREAT · SCIENTIA

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)

Augest 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 (MUCAS) 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 MUCAS 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
Mrs. Ooi, Phaik Kim
Miss Phuah, Phaik Lean
Mr. Wu, Chiaw Ching
Mr. Du, Yuchun
Mr. Walt Abell
Mr. Giovanni Moretti
Mr. John Holley

LIST OF FIGURES

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**

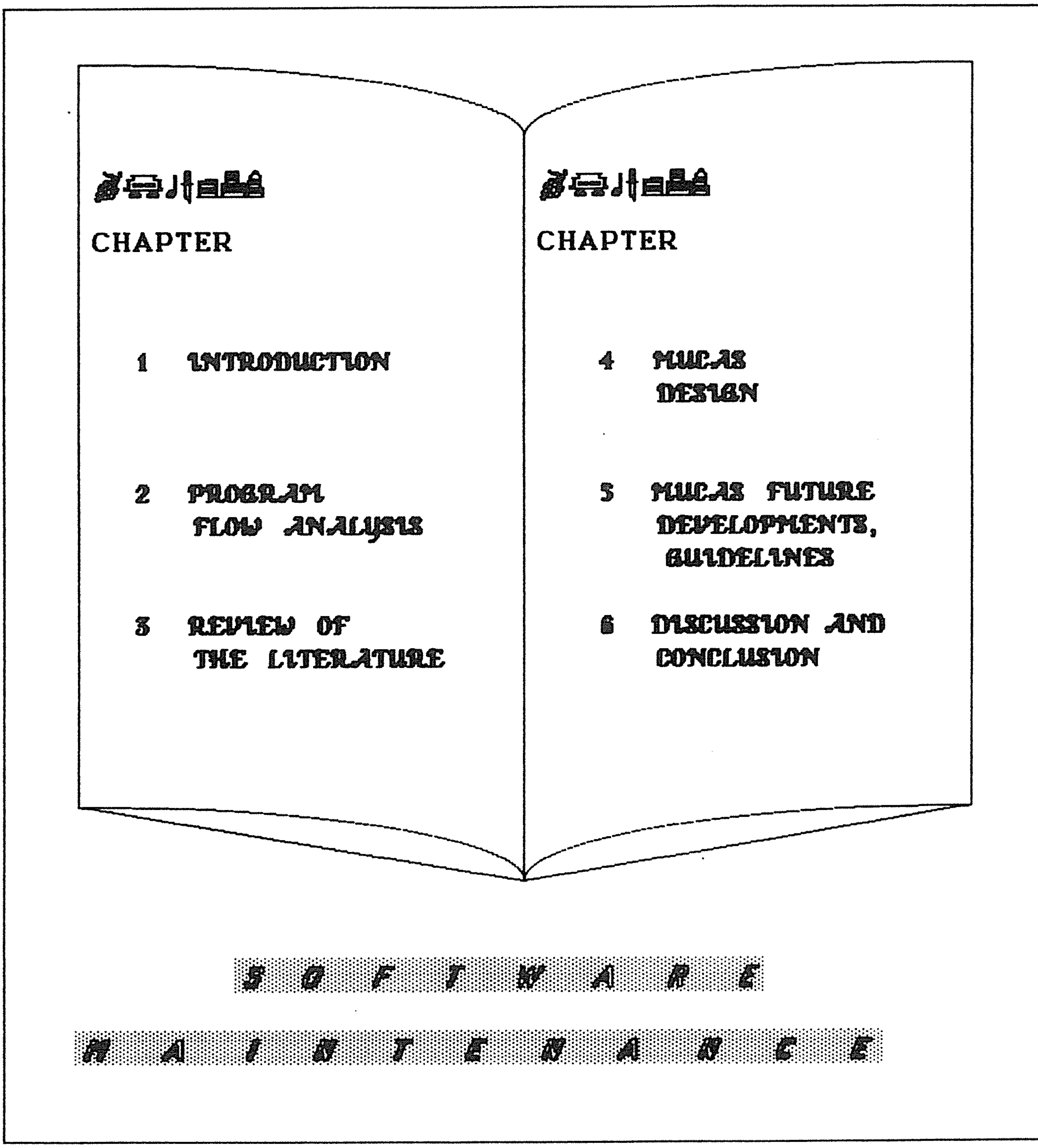

Figure P.1 What This Thesis is About.

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 <u>ad hoc retest</u> technique which uses <u>graph theory</u> [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.

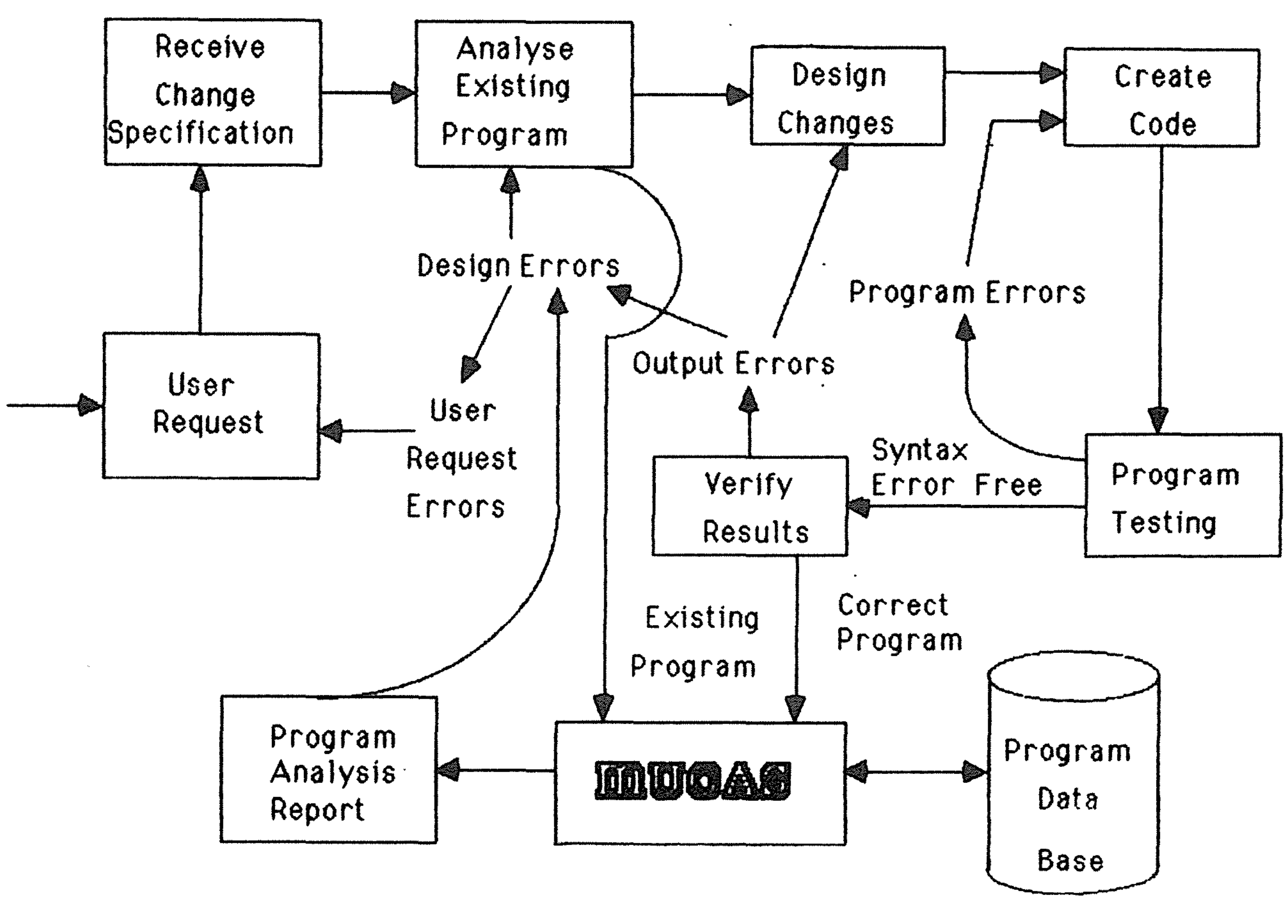

Figure 1.1 The COBOL Program Processing Activities In a MUCAS Aided Production Project.

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.