UNIVERSITI PUTRA MALAYSIA

INCORPORATING SOFTWARE MEASUREMENT INTO A COMPILER

RAFA ELAYYAN JAMIL AL QUTAISH

FSAS 1998 3

INCORPORATING SOFTWARE MEASUREMENT INTO A COMPILER

By

RAFA ELAYYAN JAMIL AL QUTAISH

Thesis Submitted in Fulfilment of the Requirements for the Degree of Master of Science in the Faculty of Science and Environmental Studies Universiti Putra Malaysia

June 1998

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

ٱقْرَأْ بِٱسْمِ رَبِّكَ ٱلَّذِى خَلَقَ ﴿١﴾ خَلَقَ ٱلْإِنسَـٰنَ مِنْ عَلَقٍ ﴿٢﴾ ٱقْرَأْ وَرَبُّكَ
ٱلْأَكْرَمُ ﴿٣﴾ ٱلَّذِى عَلَّمَ بِٱلْقَلَمِ ﴿٤﴾ عَلَّمَ ٱلْإِنسَـٰنَ مَا لَمْ يَعْلَمْ ﴿٥﴾

(In the name of Allah, Most Gracious, Most Merciful)

1. *Proclaim! (or Read!) In the name of thy Lord and Cherisher, Who created,*
2. *Created man, out of a (mere) clot of congealed blood,*
3. *Proclaim! And thy Lord is Most Bountiful,*
4. *He Who taught (the use of) the Pen,*
5. *Taught man that which he knew not.*

Surat Al-Alaq (The Clot)

الـى الذيـن من اقترن الاحسـان اليهـم بعبـادة اللـه .....
الـى الذيـن من بجبهـم تكـون القـربى والطمأنينـة والرضـوان ...
الـى مدرستـي الاولـى وكـل حياتـي .....

إليكم جميعاً أهدي رسالتي هذه...

ACKNOWLEDGEMENTS

In the name of Allah, Most Gracious, Most Merciful

I would like to take this opportunity to convey my sincere thanks and deepest gratitude to my supervisor Dr. Abdul Azim Abd. Ghani who introduce me to the field of Software Engineering. Not forgetting my co-supervisors Dr. Ramlan Mahmod and Dr. Md. Nasir Sulaiman. I am very grateful to all of them for all the help and invaluable guidance, fruitful discussions, patience and continued encouragement provided to me at every stage of this thesis. I am really touched by Dr. Abdul Azim and indeed grateful to him for having taken so much of his valuable time for studying, correcting and restructuring the preliminary drafts.

I would like to convey my appreciation to the Department of Computer Science, the University Library, the Graduate School Office, and the Laboratories Technicians.

Not forgetting to convey my sincere thanks and deepest gratitude to Prof. Mohammed Adnan Al-Bakhit, the president of Al al-Bayt University, and to all of my friends in Al al-Bayt University,

specially, Abdallah Al-Khaldi and Sa'ad Bani Moh'd for their encouragement and support all the time. In addition, I am grateful to the Arab Student Aid International (ASAI), New Jersey, U.S.A for giving me the scholarship.

I am also very grateful and wish to thank my Jordanian friends in UPM: Saleh Al-Khawaldeh, Faraj Ahmad Faraj Abu-Ilaiwi, Hasan Al-Omoush, Isam Qudsieh, Khaleel Al-Hassan, Hamed Al-Fawareh, Eid Al-Ziod, Jehad Al-Khaldi, Mohammad Saleh, Ahmad Al-Khawaldeh, Malik Al-Kedah, and Ahmad Zahran for their encouragement.

Finally, I would like to express my most sincere and warmest gratitude to my father, mother, brothers; Khaled, Jamil, Mohammad, Ali, Ahmad, and Abdullah, sisters, uncles, aunts, and cousins for their prayers, love, generous moral and financial support during my study. Also, I would like to thank my best friend Nayef Ali Al-Joulan.

TABLE OF CONTENTS

		Page
ACKNOWLEDGEMENTS	.....	iii
LIST OF FIGURES	.....	viii
ABSTRACT	.....	xiii
ABSTRAK	.....	xv

CHAPTER

I	**INTRODUCTION**	1
	Software Measurement Background	1
	Scope of the Research	5
	Objectives of the Research	6
	Organisation of the Thesis	8
II	**ISSUES ON SOFTWARE PRODUCT METRICS** .	10
	Introduction	10
	Design Metrics	11
	Information-Flow Metrics	12
	Call Graph Metrics	17
	Source Code Metrics	20
	Source Lines of Code Metrics	21
	McCabe's Metric	22
	Halstead's Metrics	28
	Automated Data Collection Tools	32
	Common Metrics Format	36

III	**ATTRIBUTE GRAMMARS: FUNDAMENTALS AND APPLICATIONS**	38
	Introduction	38
	Attribute Grammars fundamentals	38
	Attribute Grammars Applications	48
	Use of Attribute Grammars in Compiler Construction	49
	Use of Attribute Grammars in Source Code Metrics Definitions	51
	Conclusion	56
IV	**DESIGN OF THE PROPOSED COMPILER**	57
	Introduction	57
	Lexical Analyser	59
	Syntax Analyser	65
	Symbol Table Structure and Implementation	74
	Memory Allocation	88
	Code Generation	91
	Design and Implementation of the Source Code Metrics	106
	Halstead's Metrics	107
	McCabe's Metric	122
	Call Graph's Metric	125
V	**RESULTS AND DISCUSSION**	130
	Introduction	130
	Case Study 1	131
	Case Study 2	135
	Case Study 3	142
VI	**CONCLUSION AND FURTHER RESEARCH**	148
	Conclusion	148
	Further Research	150
REFERENCES		151

APPENDIX

A	Context-Free Grammar of the Subset of Pascal Language	157
B	List of Operator Names	160
C	*LEX* Implementation of the Proposed Compiler	161
D	*YACC* Implementation of the Proposed Compiler	168
E	Contents of y.tab.c	185

VITA 186

LIST OF FIGURES

Figure		Page
1	Example of Information-Flow Metrics	15
2	An Example About One of the Problems Defined by Shepperd	16
3	A Call Graph	18
4	A Call Graph and Its Metrics Calculation	19
5	Sample Pascal Codes and the Values of SLOC Metrics	23
6	Sample Source Codes and Their Control Graphs ...	25
7	Sample Pascal Code with two Possible Control Graphs	27
8	A Token Analysis of a Sample Pascal Code	30
9	An Automated Data Collection Scheme	33
10	Parse Tree for the String *xxxyyyzzz*	41
11	Parse Tree for the String *xxxyyyzzz* Using Synthesised Attribute	43
12	Parse Tree for the String *xxxyyyzzz* Using Inherited Attribute	46
13	A Diagram Illustrating the Steps in Implementing the Proposed Compiler with Software Metrics Evaluation	58
14	Interaction of Lexical Analyser with Parser	59
15	Creating a Lexical Analyser with *Lex*	60
16	The Definition of the Reserved Words Table	63

17	The *screen()* Routine	64
18	The *s_lookup()* Routine	64
19	Creating an Input / Output Translation with *yacc* .	68
20	The *yyerror()* Routine	70
21	The *yywhere()* Routine	71
22	The Contents of the Symbol Table Definitions	75
23	The Dynamic Structure of the Symbol Table	77
24	The *s_create()* Routine	77
25	The *s_find()* Routine	79
26	The *link_parm()* Routine	80
27	The *make_parm()* Routine	81
28	The *make_var()* Routine	83
29	The *blk_push()* and *blk_pop()* Routines	84
30	The *make_proc()* Routine	85
31	The *chk_parm()* Routine	86
32	The *chk_var()* Routine	87
33	The *chk_proc()* Routine	88
34	The *all_parm()* Routine	90
35	The *all_var()* Routine	91
36	Header File for the Code Generation (*codegene.h*) ..	93
37	The *gen_alu()* and *gen_li()* Routines	96
38	The *gen_mod()* and *gen()* Routines	97

39	The *gen_jump()*, *new_label()*, and *format_label()* Routines	99
40	The *gen_label()* Routine	100
41	The *push_break()*, *push_continue()*, *pop_break()*, and *pop_continue()* Routines	103
42	The *push()* and *pop()* Routines and the Definition of *bc_stack* Data Structure	104
43	The *gen_call()* Routine	105
44	The *gen_entry()* Routine	106
45	Concatenation of Lists of Identifiers	112
46	The *concate_id()* Routine	113
47	Example of the Appending of *ids* and *newlocs*	115
48	The *number_of_operators()* Routine	117
49	The *number_of_identifiers()* and *number_of_numbers()* Routines	118
50	The *total_operators()* Routine	118
51	The *total_identifiers()* and *total_numbers()* Routines	119
52	The Implementation of a Part of McCabe's Metric	123
53	The *print_result()* Routine for McCabe's Metric	124
54	A Sample of the Linked List Used to Store the Module's Names and the Names of the Called Modules	125
55	The *insert_calling_module()* Routine for the Call Graph's Metric	127
56	The *insert_called_module()* Routine for the Call Graph's Metric	128

57	The *compute_and_print_results()* Routine for the Call Graph's Metric	129
58	A Diagram Illustrating the Input / Output Structure for the Proposed Compiler	131
59	Case Study 1, Subset of Pascal Language Program	132
60	The Code Generation Produced for the Case Study 1	132
61	The Halstead's Metrics Produced for the Case Study 1	133
62	The McCabe's Metric Produced for the Case Study 1	133
63	The Call Graph's Metric Produced for the Case Study 1	134
64	Case Study 2, Subset of Pascal Language Program	136
65	The Code Generation Produced for the Case Study 2	139
66	The Halstead's Metrics Produced for the Case Study 2	140
67	The McCabe's Metric Produced for the Case Study 2	141
68	The Call Graph's Metric Produced for the Case Study 2	141
69	Case Study 3, Subset of Pascal Language Program	142
70	The Code Generation Produced for the Case Study 3	144

71 The Halstead's Metrics Produced for the Case Study 3 146

72 The McCabe's Metric Produced for the Case Study 3 147

73 The Call Graph's Metric Produced for the Case Study 3 147

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfilment of the requirements for the degree of Master of Science.

INCORPORATING SOFTWARE MEASUREMENT INTO A COMPILER

By

RAFA ELAYYAN JAMIL AL QUTAISH

June 1998

Chairman : Abdul Azim Abd. Ghani, Ph.D.

Faculty : Science and Environmental Studies.

In the area of software engineering, software measurement is not new, it was around 26 years since Halstead originally proposed a family of software measures, collectively known as software science. The magnitude of costs involved in software development and maintenance magnifies the need of a scientific foundation to support programming standards and management decisions by measurement.

This research aims at developing a compiler for a subset of Pascal language in which an evaluation for a number of software metrics has been incorporated. *Lex* and *Yacc* have been used to

generate the lexical analyser and syntax analyser for the proposed compiler. While the other components of the compiler and the metrics evaluation routines have been written in C language. The proposed compiler was implemented under *Linux* operating system. Three metrics have been incorporated to the proposed compiler, which are: Halstead's metrics, McCabe's metric, and Call-Graph metric. The software metrics will be produced in the common metrics format, which is used in SCOPE project.

Attribute grammars have been used to build the proposed compiler to evaluate the software metrics in the parsing time of the compilation process and to use a well-defined approach to the software metrics evaluation process.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Master Sains.

MENGAPLIKASIKAN PENGUKURAN PERISIAN KE DALAM SATU PENGOMPIL

Oleh

RAFA ELAYYAN JAMIL AL QUTAISH

Jun 1998

Pengerusi : Abdul Azim Abd. Ghani, Ph.D.

Fakulti : Sains Dan Pengajian Alam Sekitar.

Dalam bidang kejuruteraan perisian, pengukuran perisian bukannya sesuatu yang baru, ianya telah wujud 26 tahun yang lepas, semenjak Halstead mencadangkan satu kumpulan ukuran perisian, dikenali sebagai 'software science'. Jumlah kos yang terlibat dalam pembangunan dan penyelenggaraan perisian mewujudkan keperluan untuk mengadakan satu asas saintifik demi menyokong piawaian dan keputusan pengurusan melalu pengukuran.

Kajian ini bertujuan untuk membangunkan satu pengompil bagi satu subset bahasa Pascal dimana penilaian untuk sejumlah perisian metrik telah dikaitkan. *Lex* dan *yacc* telah digunakan

untuk menjana penganalisa leksikal dan sintak bagi pengompil yang dicadangkan. Sementara itu, komponen- komponen lain pengompil dan rutin penilaian metrik ditulis dalam bahasa C. Pengompil yang dicadangkan telah diimplementasikan di bawah sistem operasi *Linux*. Tiga metrik telah dikaitkan bagi pengompil yang di cadangkan, iaitu metrik Halstead, metrik McCabe dan metrik Call Graph. Metrik-metrik tersebut software akan dihasilkan dalam format metrik yang umum di mana ia digunakan dalam projek SCOPE.

Nahu atribut telah digunakan untuk membina pengompil yang dicadangkan supaya metrik perisian dapat dinilai dalam masa pengenal bagi proses pengkompilan dan mengunakan pendekatam terperinci dengan sewajarnya bagi proses penilaian metrik perisian.

CHAPTER I

INTRODUCTION

Software Measurement Background

Measurements have a long tradition in natural sciences. At the end of the last century the physicist, Lord Kelvin, formulated the following about measurement (Pressman, 1987):

> *"When you can measure what you are speaking about, and express it into numbers, you know some thing about it. But when you can not measure it, when you can not express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to stage of science."*

Scientists who treat with measurement theory also support this view of the application measurement in sciences. Roberts (1979), points out in his book about measurement theory:

> *"A major difference between a 'well-developed' sciences such as physics and some of the less 'well-developed' sciences such as psychology or sociology is the degree to which things are measured."*

In the area of software engineering, the concept of software measurement or what is called software metrics is not new. It was around 26 years since Halstead originally proposed a family of software measures, collectively known as software science (Halstead, 1972). The magnitude of costs involved in software development and maintenance magnifies the need of a scientific foundation to support programming standards and management decisions by measurement. Already in 1980, Curtis (1980) pointed out:

> *"Rigorous scientific procedure must be applied to studying the development of software systems if we are to transform programming into an engineering discipline. At the core of these procedures is the development of measurement techniques and the determination of cause effect relationships."*

The definition of a measure is an empirical objective assignment of a number or symbol to an entity to characterize a specific attribute (Fenton, 1991). Moreover, Ince *et al.* defined the software metrics as a numerical values of quality which can be used to characterize how good or bad that the product is in terms of properties such as its proneness to error.

Fenton (1991) has classified the software metrics to *product* metrics, *process* metrics, and *resource* metrics. In fact, he has defined the three classes as:

- *Products*: are any artifacts, deliverables, or documents that are got out of the processes.
- *Processes*: are any software-related activities.
- *Resources*:

Any thing that we are ever likely to want to measure or predict in software is an attribute of some entity of the three classes (product, process, or resource metrics). Fenton (1991) has made a distinction between attributes, which are *internal* or *external*. Internal attributes of a product, process, or resource are those that can be measured totally in terms of the product, process, or resource itself. Whereas, the external attributes of a product, process, or resource are those which can only be measured with respect to how the product, process, or resource relates to its environment.

Software metrics help in two ways. First, they help individual developers understand what they are doing and provide insight into areas that they might improve. For example, measurements of code complexity give information about which code is over-complex and might be improved by additional modularization. Measurements of

numbers and types of bugs give information on what errors a developer is prone to make, and thus what he should be watching out for. Second, software metrics gives an organization information about where it is, and about the effect of things it is trying to use (Shorp, 1993).

Grady and Caswell (1989) have summarized the advantages of software metrics. They determined that software metrics help the developer to:

- Understand software development process better.
- Measure progress.
- Provide common terminology for key controlling elements of the process.
- Identify complex software elements.
- Make software management more objective and less subjective.
- Enable the engineers and manager to estimate and schedule better.
- Better evaluate the competitive position.
- Understand where automation is needed.
- Identify engineering practices, which lead to highest quality and productivity.
- Make critical decisions earlier in the development process.

- Eliminate fundamental causes of defects.
- Encourage the use of software engineering techniques by the engineers and managers.
- Encourage the definition of long-term software development strategy based upon a measured understanding of current practices and needs.
- Be more competitive.

Scope of the Research

The research in this thesis aims at developing a compiler for a subset of Pascal language in which an evaluation for a number of software metrics has been incorporated. *Lex* and *Yacc* tools (Mason and Brown, 1990) have been used to generate the lexical analyser (scanner) and the syntax analyser (parser) for this system. While the other components of the system and the metrics evaluation routines have been written in C language. All of the system was implemented under *Linux* operating system.

The proposed compiler evaluates three metrics, which are: Halstead's metrics, McCabe's metric and Call Graph's metric. The system produces four files, three of them contain the metrics

evaluation for the three metrics (Halstead, McCabe, and Call Graph), and these files will be in common metrics format, which is used in SCOPE (SCOPE, 1991) project. And the other file will contain the intermediate code, which is generated by the system.

Attribute grammar approach has been used in this research to evaluate the software metrics in the parsing time of the compilation process and to use a well-defined approach to software metrics evaluation process.

The evaluation of the software metrics in this system is optional. However, a user can determine which metrics he wants to evaluate. Actually, he can evaluate Halstead's metrics, McCabe's metric, Call Graph's metric, all of these metrics, or none of them.

Objectives of the Research

Most approaches to software metrics have normally been used on the models of the software. Examples of the models are control flow graph and call graph, which are obtained by mapping the source code to the respective models. The models capture the relevant aspects of the source code, and from the models, software metrics are evaluated. A problem with model-based approaches is

that although the metrics are correctly evaluated from the models, the mapping from source code to the models are not precisely stated. Hence, might produce different values for the same metrics when applied to a piece of source code. Different persons may produce different models for the same piece of source code.

As an alternative for the above approaches, this research advocates an approach to software metrics based on the measurement on the source code itself. Although many software documents written in languages with well-defined syntax and semantics, the well-developed theory of programming languages is not often used as a basis for software metrics evaluation. Especially, attribute grammar formalism has been used in this research (Abd Ghani, 1996).

Building metrics evaluation into a compiler can be a good idea because of the following reasons:

- Some compilers often extend the language syntax beyond the standard thus causing external compiler-independent collection tools have difficulties coping with this.
- Some problems may be encountered in analysing source code due to missing or different include files or compilation options.

- Analysing source code duplicates the parsing of syntax, which has already done by a compiler.
- It is easier to promote the use of metrics to software developers if existing compiler can perform metrics evaluation. It reduces the cost of the software project. The software developers have not to purchase another tool to perform metrics evaluation.

Organisation of the Thesis

Chapter two gives some background on software product metrics; its fundamentals and classifications such as design metrics and source code metrics. In addition, this chapter discusses software metrics evaluation tools, utilized by previous researchers in this area. The definition and advantages of common metrics format can be found at the end of this chapter.

Chapter three shows the attribute grammars; its fundamentals and applications, mainly the use of attribute grammars in compiler construction and software metrics definitions. At the end of this chapter our conclusion will be discussed. Chapter two and chapter three are the keys of this thesis.