UNIVERSITI PUTRA MALAYSIA

DESIGNING A SOFTWARE MAINTENANCE SYSTEM USING A REVERSE ENGINEERING APPROACH

HAMED JASEM AL-FAWAREH

FSAS 1998 35

DESIGNING A SOFTWARE MAINTENANCE SYSTEM USING A REVERSE ENGINEERING APPROACH

By

HAMED JASEM AL-FAWAREH

Dissertation Submitted in Fulfillment of the Requirements for the Degree of Master of Science in the Faculty of Science and Environmental Studies University Putra Malaysia

February 1998

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

يَرْفَعِ ٱللَّهُ ٱلَّذِينَ ءَامَنُوا۟ مِنكُمْ وَٱلَّذِينَ أُوتُوا۟ ٱلْعِلْمَ دَرَجَٰتٍ ۚ
وَٱللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ ﴿١١﴾

Surat Al-Mujadilah (The Disputation), Ayah 11.

To My Late Father's Pure Spirit

ACKNOWLEDGMENTS

In the name of Allah, the Beneficent, the Merciful.

I would like to take this opportunity to convey my sincere thanks and deepest gratitude to my supervisor Dr. Abdul Azim Abdul Ghani. I am very grateful to him for the help and invaluable guidance, fruitful discussions, patience and continued encouragement provided to me at every stage of this thesis.

It is also a great honor and pleasure to acknowledge Dr. Ramlan Mahmod and Dr. Ali Mamat, members of the supervising committee, for their technical support, helpful suggestions and insight.

In preparing this thesis, a number of individuals have provided helpful for their suggestions and comments, all of which have been of tremendous help towards the compilations of this thesis. I would like to convey my appreciation to the Department of Computer Science, the University Library and Universiti Putra Malaysia for providing assistance at one time or other. I also, wish to thank all postgraduate students in the Department of Computer Science.

I am very grateful and wish to thank all my friends from Jordan, especially, Khalid Al-Tahat, Ibrahim Al-Atoum, Eid Al-Zyoud, Ziad Abu Gadora and Jehad Al-khaldi for their encouragement, help and support. Also, I would like to single out my friend Idi Fulayi to thank him for his help, encouragement and support. Also, I would like to thank Dr. Adam Kilicman for his help and encouragement.

Finally and most important, I would like to express my most sincere and warmest gratitude to my mother, eldest brother Ali, brothers; Khalid, Mammdoh, Mobarak, Omar, and Mohammed, sisters, nephews, nieces, uncles, aunts, and cousins for their prayers, love and generous moral and financial support during my studies.

All praises for the Almighty, without whose will everything would cease to be.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

$\forall$	universal quantifier
$\exists$	existential quantifier
$\wedge$	And
$\vee$	Or
$\sim$	Not
$\rightarrow$	Implication
$\leftrightarrow$	equivalence

LIST OF THE RESERVED WORDS IN THE SYSTEM

Program	Used when the identifiers scopes are all the program.
Function	Used to obtain the statistics for the functions.
Parameter	Used to obtain statistical account of all parameters in any function.
Variable	Used to give a statistical account of all variables.
And	The system knows this word as the logic "and" symbol.
Or	The system knows this word as the logic " or " symbol.
not	The system knows this word as the logic "not" symbol.

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

DESIGNING A SOFTWARE MAINTENANCE SYSTEM USING A REVERSE ENGINEERING APPROACH

By

HAMED JASEM AL-FAWAREH

February 1998

Chairman: Abdul Azim Abd Ghani, Ph.D.

Faculty: Science and Environmental Studies

The aim of the software maintenance is to maintain the software system in accordance with advancement in software and hardware technology. There are four activities to software maintenance, namely corrective maintenance, adaptive maintenance, perfective maintenance, and preventive maintenance. Three approaches are used in software maintenance, that is restructure, reverse engineering, and reenginering.

Reverse engineering is a process that is currently being used in software maintenance to extract items of information on software products. This research aims to produce a new reverse engineering logic-form tool, to help maintainers by giving them a complete document about the software system. Currently, many tools are used in reverse engineering, but not all can satisfy the problems faced by the users.

The thesis describes the design and implementation of an automatic aid system that uses a logic-form reverse engineering approach, with a new data structure called tree module structure. This tool contains four modules, namely, lexical analyzer module, syntax analyzer module, handling module, and interface or query module.

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

MEREKABENTUK SISTEM PERISIAN PENYELENGGARAAN DENGAN MENGGUNAKAN PENDEKATAN PEREKASAYAAN SONGSANG

Oleh

HAMED JASEM AL-FAWAREH

February 1998

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

Fakulti: Sains dan Pengajian Alam Sekitar

Matlamat penyelenggaraan perisian adalah untuk menyelenggara sistem perisian supaya menepati dengan kemajuan dalam teknologi perisian dan perkakasan. Terdapat aktiviti dalam penyelenggaraan perisian iaitu penyelenggaraan pembetulan, penyelenggaraan penyesuaian, penyelenggaraan penyempurnaan dan penyelenggaraan pencegahan. Tiga pendekatan digunakan dalam penyelenggaraan perisian iaitu penstrukturan semula, perekasayaan songsang, perekasayaan semula.

Perekasayaan songsang adalah suatu proses yang digunakan dalam penyelenggaraan perisian untuk mengeluarkan butiran maklumat mengenai produk perisian. Penyelidikan ini bertujuan untuk menghasilkan satu peralatan baru perekayasaan semula berbentuk-logik untuk membantu penyelenggara dengan menyediakan mereka satu dokumen lengkap mengenai sistem perisian. Sekarang ini banyak peralatan

digunakan dalam perekayasaan semula tetapi tidak semua dapat memenuhi masalah yang dihadapi oleh pengguna.

Tesis ini menerangkan rekabentuk dan implementasi sistem bantuan automatik yang menggunakan pendekatan perekayasaan semula berbentuk logik dengan kaedah baru yang dipanggil modul pokok sistem pendekatan. perekayasaan semula berbentuk logik menggunakan empat modul iaitu modul penganalisis leksikal, modul penganalisis sintak, modul pengendalian dan modul antara muka atau pertanyaan.

CHAPTER I

INTRODUCTION

Background

Computer systems have been used in many complex and diverse applications; they are also applied in critical areas, where the useful of a nation may depend on them. These computer applications increased with the advance in technology, also, the complexity has increased a lot in these applications. Each software system may contain thousands of components which, may be spread over different and large modules. The relationships between these components make determination of any change between the diverse parts of the software system very difficult. This problem becomes more complicated for a large software application. The integration of these modules becomes tedious and time consuming without the support of aid tools.

During the initial years of computer software and hardware development programmers concentrated on the production of new software systems for new applications. These software systems consumed much time and money for the development.

In the early 1960s, programmers attention was directed towards in producing new applications, rather than updating the old software. In the 1970s, the developments in computer software and hardware technology caused programmers to maintain the old

software and increase the life cycle of software systems. Furthermore, attempts to produce new software during this period was time consuming and costly, so maintainers tended to maintain the software system by using software maintenance (Glass & Noleseux 1981).

It is easy to detect a logic error when a maintainer builds a compiler e.g. in C-language when the maintainer forgets a semicolon at the end of a line or omits to close brackets in a "for statement". The main problem is that the maintainer finds it more difficult to detect run time errors inside the program. These errors cause the program to produce incorrect results.

Detecting and correcting errors take much time. For example, in a given function, a maintainer needs to know all the input and output parameters. To solve these problems; he needs to trace all the variables and functions called during the execution of a program. In order to do this, the maintainer needs a list of all local and global variables, and function calls, to determine the statements that contain errors.

Errors in a complex software system are difficult to correct and detect; software maintenance aids the maintainer to correct and trace the software system, and, thus, reduces its cost. When programmers build software systems, the task of detecting and correcting errors for small software system is simple; but it becomes difficult and complex for large software systems. One of the software maintenance activities is concerned with correcting software errors and producing full details about the components of the program, such as, global variables and local variables.

Software maintenance activities modify old software systems, correct errors in order to improve the performance of these systems, and adapt the software systems to the changing environment. There are four types of software maintenance activities. First, corrective maintenance, which deals with testing of large, and huge software systems, to detect errors, which may need much time to correct as whole contents of the program need to be traced. The second is adaptive maintenance. This looks into the rapid change in software environment, such as the operating system, new hardware, and upgrading of the elements of the system. The functionality of the software however does not change. The third which, is perfective maintenance deals with the modification and implementation of the recommendations from users, after the use of the software. The fourth is preventive maintenance which brings about future enhancement in the software system.

In general, software maintenance is very expensive compared to software development. The cost of software maintenance became higher after 1970s, and by the 1990s, it constituted more than 62% of the cost (Zuylen, 1993). Three approaches are used to maintain software. These are the 3RE's namely, reengineering, restructuring, and reverse engineering (Chikofsky, et. al., 1990). Reengineering is concerned with processing the existing software by producing a new source code without changing the old system function. Restructuring involves examining the existing software and rewriting parts of it by changing unstructured, ambiguous and difficult software to a new structure, that is easier to understand. Reverse engineering is a technique to analyze a subject system to identify system components and their relationships, in addition to

creating representations of the system at a higher level of abstraction. In other words, reverse engineering is a process of extracting information from a source code concerning software product design.

In this thesis, the reverse engineering approach will be used to produce a new tool for helping users to walk through the software system. This system is called "logic form reverse engineering tools". This system is divided into four modules, namely, lexical analyzer module, syntax analyzer module, handling module, and interface module.

Logic Form Reverse Engineering Approach

When a developer builds a huge and large software system, detecting and correcting errors is time consuming. For example, for a function of a given program, the developer may wish to know all the input and output parameters, these parameters help the developer to correct software errors. If it is a small program, errors are easy to detect and correct. However, it becomes difficult and complex in large software.

Software systems are spread over different modules. These modules contain numerous local and global variables, parameters, and functions, in different level of abstractions. Program construction in a sub standard C-language is spread over different functions. Not only do these functions contain separate variables but also the program itself contains global and local variables. The overlap between all functions, local, and global variables, makes the walking through inside the program more difficult if it is

done manually by the user. The main attention of users is to detect all variables and functions declared in software system in order to correct errors.

The errors that occur in a software system are of three kinds: implicit error, latent error, and syntax errors or the easy errors. Syntax errors are the most common. They occur when the user, for example, forgets the semicolon at the end of line, or forgets to close the bracket in a "for statement". This error is very easy to detect during execution of a software system. Implicit error or run time error is difficult to detect because the system compilation does not detect it. Also, this error may feature incorrect results after execution of the program. To detect this error, the maintainer needs to walk through the program and trace all the variables and functions declared in the program. This method is easy in small programs, but becomes complex when the software system grows large and huge. When the users use the software system, they write full recommendations to the maintainer in order to change the software and correct the error, again, software system need to do program tracing.

For the above problems we developed a logic form reverse engineering tool. This tool contains four components, namely: LEXICAL ANALYZER, SYNTAX ANALYZER, HANDLING MODULE, and QUERY MODULE. Logic-form reverse engineering tool assists in enhancing system constituents and their relationships. Also, it assists in binding the modules and variables in the program. It also helps the user to recognize the overlap between modules and variables in the program, and identifies components and their relationships. Furthermore, it provides information that helps users

to correct errors in the program. This tool helps users to walk through all contents of the program, its functions and its variables without encountering any difficulty for correcting them. Also, it detects implicit faults so as to get the correct results.

Aims of the Research

The research in this thesis, aims at developing a logic form reverse engineering system. It is designed to help a maintainer correct the three kinds of errors, by giving a full documentation for all local and global variables, parameters, and functions used in the software system. In addition, this system extracts information concerning software variables and functions and provides information on call graph. This information should help to control many managerial problems. These problems may occur when building large software systems for a large number of families of components. For example, let us consider a database containing thousands of components or modules and thousands of dependency relationships, and documents. It is beyond the human ability to search manually such a database to find specific information about specific modules, their dependencies, and connectivity to other activities in the life cycle of a product. This system provides all the facilities for browsing the database, and gives all the information about the components of software systems, such as, global variables, local variables, functions, and parameters. Furthermore, this tool clears overlapping between the software system components, variable and functions.

The system developed in this research provides visibility to maintainers, by allowing them to go through the life cycle of the functions and variables, and increasing

the understandability of the software system. Also, the research should help maintainers to correct programming errors, and achieve more correct results. This system, also, provides querying and browsing facilities.

Organization of the Thesis

Chapter two provides background information on software maintenance such as its fundamentals, activities and approaches, which are the focus of this thesis. In addition, this chapter discusses reverse engineering tools, utilized by previous researches in this area. Chapter three describes the methodology used in the logic form reverse engineering approach. Chapter four describes the design and implementation of a logic-form reverse engineering system. This system contains four modules: lexical analyzer module, syntax analyzer module, handling module, and interface module. Furthermore, it describes the method which, is used in this system that is "tree module". Chapter five discusses the result of the implementation of the logic-form reverse engineering system as well as the sample of the output, that is capable of responding to certain maintainer queries. Chapter six provides a conclusion of the approach discussed in this thesis and evaluates the achievements of this approach. Some suggestions for further work are also discussed at the end of the chapter.

CHAPTER II

SOFTWARE MAINTENANCE: ACTIVITIES AND APPROACHES

Introduction

This chapter provides an overview of software maintenance system and its relationship to the 3REs: Restructuring, Reengineering, and Reverse Engineering. It also explains various reverse engineering tools.

Software Maintenance

Emergence of Software Maintenance

During the period between 1950s and early 1960s, software maintenance was restricted to a very small part of the software life cycle. The main activity of the programmer was to write new programs for new applications. By the end of the 1960s and early 1970s, most software systems that had been written had become obsolete. However, it was very difficult and costly to get rid of all these software system. It was then that software maintenance emerged. Software system life cycle history can be categorized in two stages: the development and the maintenance stages. During the development stage, the life cycle was used as a development model whereby development activities spread in a sequential pattern, beginning from the requirements,