AN APPROACH TO SOFTWARE MAINTENANCE SUPPORT USING A SYNTACTIC SOURCE CODE ANALYSER DATA BASE

This thesis is presented in a partial fulfillment of the requirements for the degree of Master of Arts in Computer Science at Massey University.

PETER VIVIAN PARKIN

1987

ABSTRACT

In this thesis, the development of a software maintenance tool called a syntactic source code analyser (SSCA) is summarised. An SSCA supports other maintenance tools which interact with source code by creating a data base of source information which has links to a formatted version of program source code. The particular SSCA presented handles programs written in a version of COBOL.

Before developing a SSCA system, aspects of software maintenance need to be considered. Hence, the scope, definitions and problems of maintenance activities are briefly reviewed and maintenance support through environments, software metrics, and specific tools and techniques examined. A complete maintenance support environment for an application is found to overlap considerably with the application documentation system and shares some tools with development environments. Program source code is also identified as the fundamental documentation of an application and interaction with this source code is a requirement of many maintenance support tools.

ACKNOWLEDGEMENTS

I wish to record my gratitude to Professor Graham Tate for his guidance and supervision of this thesis.

Also, I would like to thank :

June Verner for her interest and support in this research;

My flatmates for encouraging my endevours;

and Massey University for providing the necessary facilities required for this thesis.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1. INTRODUCTION.

This thesis is concerned with software maintenance and tools and techniques for the support of software maintenance. This chapter briefly outlines the areas covered by later chapters and their sections.

Software maintenance is an expensive area of the system life cycle consuming an estimated 32% of system costs [MCK84]. Although maintenance is now beginning to be recognised as important, the amount of direct maintenance research which has been carried out is limited. Exactly what constitutes a maintenance task is still not completely defined especially the demarcation between maintenance and redevelopment. The emphasis of Chapter 2 is on defining and describing various aspects of maintenance (particularly aspects which are considered problematic), examining the relationship between development and maintenance and attempting to identify general principles for the modification of software.

In Section 2.1 a broad definition of maintenance is given and discussed. Reasons for maintenance are also examined in this section. The reasons suggest that maintenance is fundamental to most computer systems.

Although it could be supported by general tools, like fourth generation languages (4GLs), maintenance will certainly not disappear in the future [TAT85].

A task which has been identified as maintenance can be further classified using a number of categorisation schemes. These schemes, and some of the benefits and dangers in using them, are investigated in Section 2.2.

Source code produced in maintenance costs between 10 and 100 times more than in development [CON84]. High code production costs and maintenance backlogs of up to 2.5 years [TIN84] suggest that particular problems occur in maintenance which hamper increases in productivity. Several surveys of DP managers and/or programmers [CHA85] [LIE78] [REU81] have been carried out in an attempt to identify maintenance problems. Results from these surveys and suggestions from other researchers are discussed in Section 2.3.

Section 2.4 helps to further define maintenance in terms of its place within the system life cycle. In this section, the steps or actions associated with any software modification task (i.e. maintenance task) are also identified. The definition of aspects of maintenance is completed in Section 2.5 with an examination of direct influences on the process of software modification and a description of the phenomena known as "ripple effect" and "structural decay".

Section 2.6 and part of Section 2.5 are devoted to discussing principles for achieving successful maintenance. Difficulties with identifying such principles are illustrated through the design and implementation of modifications to a particular COBOL program (the program is given in Appendix 1).

Having defined maintenance and its problems in Chapter 2, tools and techniques to support various aspects of maintenance are presented in Chapter 3. Static complexity metrics (usually applied to individual programs) have been suggested as measures of the difficulty in understanding source code in maintenance and producing debugged source code in development. These metrics are directly applicable in maintenance as the code exists whereas for most development operations they must be estimated. The metrics range from simple counts of language tokens in a program through measures requiring the application of complex algorithms for their calculation. Section 3.1 reviews and compares many proposed complexity metrics.

Various documentation is used by managers, users and maintainers to aid understanding of an application system. As well as using documents, maintenance is concerned with keeping documents up-to-date and consistent. Program source code itself is a form of documentation. Several systems or environments have been proposed for general documentation support (these are summarised in Appendix 2). Aspects of documentation support relevant to maintenance, including document categorisation, are discussed in Section 3.2.

Software tools can automate or, at least, support many maintenance related tasks including reformatting, control and data flow analysis, restructuring and dynamic analysis of programs. Such tools are often useful both in development and maintenance (e.g. RXVP [EBE80] and SADAT [VOG80]). A number of tools are briefly summarised in Appendix 2. In Section 3.3, these tools are classified and general maintenance support through tools is examined.

Syntactic analysis of a program's source code is a feature of many tools. Frequently, tool functions make use of a pool of syntactic information gathered earlier. For example, the control and data tracing features of MAP [WAR82] and program instrumentation for dynamic analysis in RXVP [EBE80]. Syntactic analysis and the production of a syntactic data base are tasks worth isolating in single purpose software tool. In Section 3.4, the idea of a program analysis system composed of a variety of tools, most of which make use of syntactic data base information, is explored. The logical contents of such a data base are also identified in this section.

Chapter 4 summarises the implementation of a Syntactic Source Code Analyser (SSCA) and it's database (SSCA DB) for a version of COBOL. Such a system is a first step toward a maintenance support system based on static analysis. Availability of a SSCA DB should encourage development of more advanced COBOL analysis tools and provide a measure of integration between these tools.

COBOL was chosen as the language to analyse because of the large number of commercial programs written in it (approximately 80% according to Al-Jarrah and Torsun [TOR79]). However, the proposed revised X3.23-Sept. 1981 COBOL language definition [COB81] defines a large and complex language composed of a nucleus and eleven functional modules. For a prototype SSCA, it was considered desirable to reduce this standard by removing many special purpose modules and simplifying some language features. The reduction process is outlined in Section 4.1 and Appendix 3A. Appendix 3B contains the reduced COBOL language definition.

Part of developing a SSCA system involves selection of an appropriate Data Base Management System (DBMS) for the SSCA DB and detailed data design for the SSCA DB. Section 3.4 has already presented a logical view of what should be in this database. In Section 4.2 this view is elaborated for application to COBOL. The new data model is then used for the selection of a DBMS (the INGRES relational system was chosen) and, finally, an implementation data model is prepared.

Section 4.3 describes the methods employed to build a syntactic analyser and formatter for COBOL programs. The implementation was carried out using a number of construction devices available on a VAX 11/750 running ULTRIX-32. ULTRIX-32 is a trademark of the Digital Equipment Corporation. The construction tools included C (a general purpose programming language), AWK (a pattern matching language), LEX (a lexical analysis preprocessor for C), YACC (a grammar parsing preprocessor for C) and EQUEL (a C/INGRES interaction language). Extensive use was also made of the technique for transferring data between executing processes known as piping.

Chapter 5 presents conclusions from the research carried out in this thesis. The conclusions cover areas such as maintenance in general, maintenance support through software tools and evaluation of the SSCA development presented in Chapter 4.