205

AN EXPLORATORY STUDY OF SOFTWARE DEVELOPMENT MEASURES ACROSS COBOL PROGRAMS

by

NADINE M. VEEDER

B.S., KANSAS STATE UNIVERSITY, 1984

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Approved by:

Major Professor

Table of Contents

List of Figures

Acknowledgements

The author wishes to express her sincere gratitude to Dr. Austin C. Melton Jr., Dr. David A. Gustafson, and the Software Measures Seminar participants for their guidance and assistance during the course of this research. A special thanks is also extended to the corportion which most gratiously supplied the data for this research effort.

The author also wishes to thank her parents Mr. and Mrs. Merton H. Veeder for their love and support through many years of education.

Chapter One

Introduction

The software development process is complex and difficult to understand. Many variables including the hardware available, the language used, the type of problem to be solved, the algorithm used in solving the problem, the methodology used in developing the software, the programming environment, and the programmer's ability affect the software development process. While many of these factors can be easily controlled, the variablity of the human element is inherent. Each programmer will approach the task of software development in a different manner, and worse yet, some programmers may unnecessarily increase the complexity of the task at hand.

Software managers have recognized the need for objective methods for monitoring the progress of software development and for anticipating problems in the development process. Software complexity measures/metrics are an attempt to satisfy this need. The terms measures and metrics are used for the most part interchangeably in the literature. The term measure will be used in this report. A Software complexity measure, in this context, is a quantification of the attributes of a program. This study quantifies the attributes of modifications made to a program during development, as such the measures described in this

report are software development measures.

Several software complexity measures have been proposed and explored. In general, these measures can be classified as one of the following: size measures, data structure measures, logic structure measures, composite measures, design measures, cost measures, or defect measures. All of these measures are plagued with being environment dependent, requiring re-calibration of parameters when transported from one environment to another (ie. historical data). Since measures are environment dependent, empirical studies and experimentation are required to assess the effects of the environment on a measure.

Even though empirical studies and experimentation are essential for the development and assessment of a software measure, the cost of the data gathering process often prevents many studies and experiments from occurring. "Real world" data is especially difficult to obtain as industry is rarely willing to expend the time, effort, and dollars to acquire the necessary data. Therefore, many studies are conducted in a university environment under controlled situations. While these studies are certainly worthwhile and commendable, "real world" data is an essential ingredient in assessing measures which will be applied to "real world" software development.

The study described in the following chapters is an exploratory study of "real world" COBOL programs in the

development phase. This study generated voluminous amounts of data in order to better understand what goes on in the software development phase and also to gain some insight into the software development process in general.

Chapter two describes the data analyzed in this study as well as the measures applied to the data, the tools which implemented these measures, and the methodology used in analyzing the data.

Chapter three describes the results of this study. This description is broken into five categories: measures on all statements in the program collection, measures on the added sections in the program collection, measures on the added statements in the program collection, measures on the changed statements in the program collection, and measures on the deleted statements in the program collection.

Chapter four includes the conclusions of this work, along with suggestions for future work.

Chapter Two

Analysis of the Data

2.1. The Data Collection

Thirty one COBOL programs in the development phase were analyzed. These programs were obtained from a Kansas corporation, and thus were written by "professional" programmers. The programs were relatively short (on average 1630 lines of code), and had varying numbers of revisions (See Figure 1.) Hereafter, the term revision will be used to refer to a particular revision of a program.

revisions	number of programs
1	6
2	3
3	7
4	7
5	3
6	3
7	1
12	1

Figure 1. Number of Revisions for Programs

Since the data gathering process was not done in a controlled environment, several difficulties arose in analyzing

the data.

1) Little was known about the steps in the software life cycle preceding the coding phase.

2) Little was known about how revisions were gathered, and furthermore there was no assurance that all revisions labeled as revision 0 were actually the first attempt at coding the program. (Investigations of the programs provided evidence to the contrary.)

3) Program collections were incomplete, and few programs with large numbers of revisions existed.

4) Structured coding techniques (including modularization) were not utilized, and therefore module by module analysis was not possible.

2.2. Measures Used to Analyze the Data Collection

Two sequential revisions of a program were analyzed using the measures described below. The result of this analysis was the quantification of the modifications made to the first revision which yielded the second revision. The measures were computed using the Data Division statements and the Procedure Division statements only. The Identification Division and Environment Division statements were excluded from this analysis due to their relative stability.

(The notation used below was devised by Dr. David A. Gustafson and the participants of the Software Measures Seminar at Kansas State University.)

TYPE: represents the type of statement being analyzed where ALL represents the collection of all statement types

Statement types were classified by their functions. The classification used is described below. (Only those statement types occurring in the data are listed below.)

TYPE ::= ALL | comments | declarations |
assignments | conditionals | branches |
input/output | labels | other

comments ::= spacing purposes | textual

assignments ::= MOVE | ADD | SUBTRACT | COMPUTE

conditionals ::= IF | ELSE | ON | AT END

branches ::= CALL | PERFORM | GOTO | NEXT | EXIT

input/output ::= DELETE | DISPLAY | OPEN | READ |
REWRITE | WRITE

other ::= EXAMINE | INSPECT | SEARCH | SORT | SET |
EXEC CICS | GOBACK

Number[TYPE]: number of statements of specified TYPE in the program.

Changes[TYPE]: number of statements of specified TYPE that have been changed.

Deletes[TYPE]: number of statements of specified TYPE that have been deleted.

Adds[TYPE]: number of statements of specified TYPE that have been added.

Add_Sections[TYPE]: number of statements of specified TYPE after which a section of code has been added.

Modification: as a notational convenience, let

Modification ::= Changes | Deletes | Adds | Add_Sections.

It is assumed that the values of Modification and TYPE are constant for an equation; for example, Modification%[TYPE] may never be Changes[IF]/Adds[ELSE].

Modification%[TYPE]: Modification[TYPE]/Number[TYPE]

%Modification[TYPE]: Modification[TYPE]/Modification[ALL]

SD_%Modification[TYPE]: standard deviation of %Modification[TYPE]

ANL: average level of nesting for a program.

Let Ni be the number of statements at nesting level i

Then ANL =

(summation (i=0 to N) of i * Ni)/Number[ALL]

ANL[TYPE]: average level of nesting of the statements of TYPE in a program.
Let Ni be the number of statements of TYPE at nesting level i
Then ANL[TYPE] =
(summation (i=0 to N) of i * Ni)/Number[TYPE]

ANL_Modification[TYPE]: ANL of Modification[TYPE]

SD_ANL_Modification[TYPE]: standard deviation of ANL_Modification[TYPE]

Number[TYPE,LEVEL]: number of statements of TYPE that are at LEVEL of nesting.

changes[TYPE,LEVEL]: number of changes of statements of TYPE that are at LEVEL of nesting.

deletes[TYPE,LEVEL]: number of deletions of statements of TYPE that are at LEVEL of nesting.

adds[TYPE,LEVEL]: number of additions of statements of TYPE that are at LEVEL of nesting

add_sections[TYPE,LEVEL]: number of statements of TYPE after which a section of code has been added that are at LEVEL of nesting.

CHANGES[TYPE1,TYPE2]: number of changed statements of

TYPE1 which changed to a statement of TYPE2

CHANGES%[TYPE1,TYPE2]: CHANGES[TYPE1,TYPE2]/Changes[TYPE1]

In addition to the above measures, one additional software development characteristic was quantified. The nesting environment of a changed or deleted IF/ELSE statement was quantified.

2.3. Rational Behind Selected Measures

The measure Modifications%[TYPE] is the percentage of statements of a specified TYPE which are being modified relative to the total number of statements of that TYPE in the program. (Here Modifications ::= Deletes | Changes) This measure indicates the relative stability of a particular statement type in a program. The program size heavily influences this measure.

The measure %Modifications[TYPE] is independent of the program size. In the case of Modification ::= Deletes | Changes, this measure shows which statement types are being modified most frequently relative to all modified statements. In the case of Modification::= Adds, this measure shows which statement types are being added most frequently. In the case of Modification::= Add_Sections, this measure depicts the most likely places for sections of code to be added.

The measure SD_%Modification[TYPE] reflects the dispersion of values for %Modification[TYPE] from the mean. If the dispersion is small and thus the numbers are grouped closely together, the measure of %Modification[TYPE] is more reliable and descriptive of the data, than if the spread is greater.

The measure ANL[TYPE] depicts the average nesting level of a statement type. This measure indicates where a particular statement type is most likely to be located.

The measure ANL_Modification[TYPE] depicts the average nesting level of statement types which are modified. This measure can indicate at which level modifications are most likely to occur.

The measure SD_ANL_Modification[TYPE] reflects the dispersion of values for ANL_Modification[TYPE].

The measures changes[TYPE,LEVEL], deletes[TYPE,LEVEL], adds[TYPE,LEVEL], and add_sections[TYPE,LEVEL] show the actual number of statements of a particular type which are modified at each level. This measure gives a clearer picture of where statements are being modified than the previous measures.

The measures CHANGES[TYPE1,TYPE2] and CHANGES%[TYPE1,TYPE2] depict the number of statements of TYPE1 which are changed to a statement of TYPE2, and the percentage of statements of TYPE1 whicanged to a

statement of TYPE2, respectively. It would appear that statements would change to either a statement of the same type or a statement of a similar type. These measures were applied to the programs in order to determine if this were the case.

The environment of a changed or deleted statement was quantified to determine if the environment of a modified statement influenced the amount of modifications occurring. Highly nested code is generally believed to add to the complexity of the source code. Intuitively, as the complexity of the source code increases, so should the number of modifications occurring. Thus, it would most likely be the case that statements in the encompassing scope of a highly nested statement will change quite frequently. To serve as a test case, the IF and ELSE statements were considered in this analysis. Even though only two statements were used in this analysis, it should be the case that any statement in the encompassing scope of a highly nested statement will change quite frequently.

2.4. Tools Developed for Analyzing the Data

A set of tools was developed to analyze the COBOL programs. This set includes preprocessing modules, a difference module, tabulation modules, and report producing modules. Figure 2 gives a hierarchy diagram of the tool set modules.

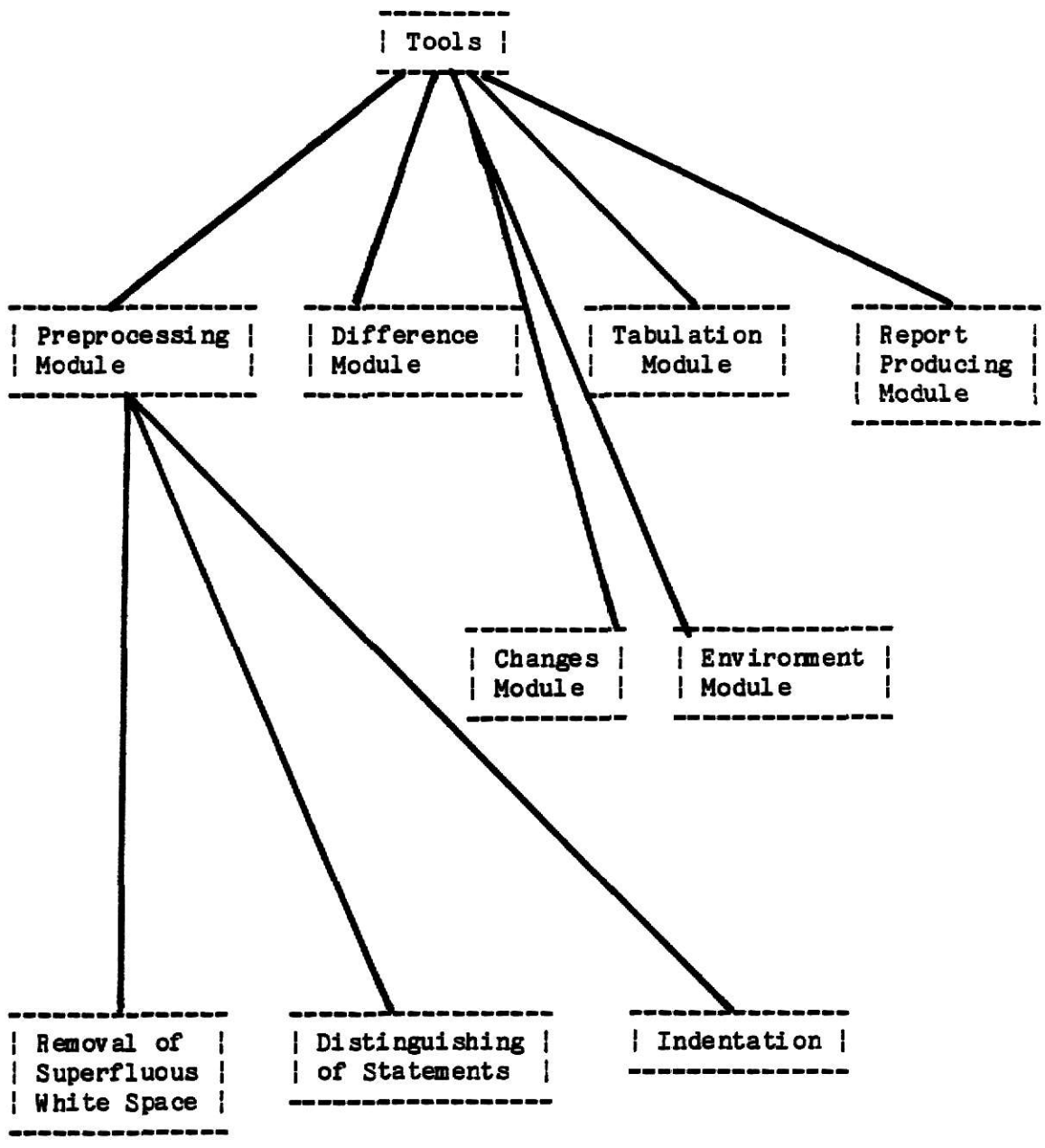

Figure 2. Hierachy Diagram of the Tool Set Modules

2.4.1. Preprocessing of Programs

Preprocessing of the programs was necessary to prepare them for the difference module. The preprocessing modules have three basic functions: 1) to remove all superfluous "white space" in the programs, 2) to make distinctions between similar statements in different Divisions of a COBOL program, and 3) to ensure uniform indentation.

2.4.1.1. Removal of Superfluous "White Space"

All superfluous blanks, tabs, and newlines were removed to ensure proper identification of a change. For example, if the only change between Statement A in revision one and revision two is the addition of a blank or a tab, Statement A has not been significantly changed. Therefore, this type of a change was ignored. Similarly, if a statement is spread over two lines in revision one and occupies three lines in revision two but otherwise is the same, no significant change has occurred. This type of change was ignored by ensuring that each statement occupies only one line.

2.4.1.2. Distinguishing of Statements

Initially, each program was searched to verify the existence of all four Divisions. (Identification Division, Environment Division, Data Division, and Procedure Division) Special flags were inserted into the programs where these

Divisions were missing. Using either the actual Divisions or the special flags as reference points, statements belonging to each of the divisions were prefixed by a unique character signifying which Division a particular statement belonged to. The prefixing of statements by unique division characters was done to ensure that each statement, when out of its context, could be properly identified. For example, if the statement "MOVE X TO Y." appeared in the Identification Division as a Remark, it should not be considered as a MOVE statement but rather as a comment. However, had preprocessing of this "MOVE" statement not occurred, later modules would not be able to make this distinction.

2.4.1.3. Indentation of a Program

A standard indentation scheme was imposed upon the programs. The statements were indented one tab character per each level with level 0, the outermost level, having no indentation. Due to the unstructuredness of the COBOL programs, only IF statements were used to determine the nesting level.

2.4.2. Difference Module

A difference module, utilizing the "diff" utility, had been developed and used previously for analyzing changes to maintenance programs written in C. This module, with minor

alterations, was used to analyze the COBOL programs.

This module compares two programs and notes changes, deletions, and added sections by prefixing statements in the first revision by special characters. This module can also be used to obtain which statements were added to the first revision by interchanging the inputs to the module.

2.4.3. Tabulation

Using the results of the difference module, four additional files are created. One file consisting of all the changed statements, one consisting of all deleted statements, one consisting of all the added statements, and one file consisting of all the statements after which a section of code was added. Each of these four files along with the entire program is used in sequence as input to the main tabulation module.

The main tabulation module counts the number of each type of statement and the number of each statement type at each level of indentation. These counts are written to a temporary file.

Upon completion of this module, five new files exist: an overall counts file, a changed statement counts file, a deleted statement counts file, an added statement counts file, and an added section counts file.

2.4.4. Report Production

Several short C and Shell programs were written to produce several different reports from the temporary files created by the tabulation module. One report was generated for each of the measures described in Section 2.2.

2.4.5. Other Modules

Two additional tools were written which utilize the output of the difference tool. One module utilizes the "diff" utility to analyze the types of statements which changed statements were changed to (ie., the resulting type of changed statements). The other module utilizes the "awk" utility to examine the environment of a changed statement.

2.5. Method of Analysis

The initial objective of this analysis was to be able to discern distinct patterns from revision to revision in program development. However, due to the problems discussed in Section 2.1, this objective could not be fully realized. Thus, the following approach in analyzing the data was adopted.

1) All programs were analyzed using the measures described in Section 1.2.

2) Programs in early phases of development were

analyzed using the measures described in Section 1.2.

3) Programs in the later phases of development were analyzed using the measures described in Section 1.2.

This approach allowed the characteristics of the entire program collection, along with the characteristics of programs in the early phases and later phases of development to be analyzed. The intent of this approach was to compare the characteristics of programs in the early phases of development with the characteristics of programs in the later phases of development, thus allowing distinctions to be made between the characteristics of programs in the early phases of revision and programs in the later phases of revision.

Programs in revisions 0 through 2 inclusive were selected as programs in the early phases of development. Programs having 6 or more revisions were selected as programs in the later phases of development. A wider variance in revision numbers were included as programs in the later phases of development to allow the collection to be more representative of the entire program collection. Unfortunately, only one program had more than 7 revisions, while 4 programs had 6 or more revisions. (See Figure 1.)

Two sets of programs were excluded from this analysis. One program set was excluded on the basis that it had no

modifications through the entire development cycle. The second program was excluded due to radical changes during development, this program doubled or tripled in size with each revision.

Chapter 3

Results of Analysis

3.0. Introduction

The results from the analysis of the data collection will be presented as follows:

1) characteristics of the programs,

2) characteristics of the added sections in programs,

3) characteristics of the added statements in programs,

4) characteristics of the deleted statements in programs,

5) characteristics of the changed statements in programs.

3.1. Program Characteristics

The program collection as a whole will be discussed in this section. The characteristics of programs in the early phases of development and later phases of development are not presented in this section; however, the characteristics may be observed in the Appendix in Figures 3-9. The characteristics of programs in the early phases of revisions and programs in the later phases of revision are not presented

as they are not necessarily indicative of any patterns of development, but rather were used as a reference point for other measurements which were applied to the data.

3.1.1. Number[TYPE] and Number[TYPE]/Number[ALL]

The entire program collection had a total of 157,575 statements in the Data Division and Procedure Division combined. 33.64% of the statements were assignment statements; 23.27% of the statements were declarations; 13.82% of the statements were conditional statements; 11.77% of the statements were comments; 10.93% of the statements were branching statements; 3.85% of the statements were labels; 2.42% of the statements were other statements; and 0.30% of the statements were input/output statements. (See Appendix Figure 3.)

The relatively high percentage of comments is rather misleading. 69.21% of the comments were used only for vertical spacing of the code. 30.79% of the comments were "textual" comments. These were comments containing some text. Unfortunately with rare exceptions, these were statements being commented out for debugging purposes.

In the category of branching statements, the unstructuredness of the code is revealed in the relatively high percentage of GOTO statements. 6.23% of all the statements were GOTO statements (9813) and 3.22% of the statements were PERFORM statements (5067). (See Appendix Figure 3.)

3.1.2. Number[TYPE,LEVEL]

The level of nesting of statements ranged from level 0 to level 7. 50.12% of the statements were at level 0; 36.49% of the statements were at level 1; 10.92% of the statements were at level 2; 1.63% of the statements were at level 3; 0.44% of the statements were at level 4; 0.19% of the statements were at level 5; 0.19% of the statements were at level 6; and 0.07% of the statements were at level 7.

Levels 0 through 2 inclusive had a wide variety of statement types. However, as the nesting level increased, the variety of statements present decreased. At level 3, eight different statement types were present, they were textual comments, MOVE statements, IF statements, ELSE statements, PERFORM statements, GOTO statements, NEXT statements, and OPEN statements. However, at level 7, only three different statement types were present: textual comments, MOVE statements, and PERFORM statements. (See Appendix Figures 6a and 6b.)

3.1.3. ANL[TYPE]

The average nesting level of the entire program collection was 0.67. The statements which had the highest average nesting level were: the OPEN statement (2.18), the ELSE statement (1.75), the PERFORM statement (1.08), textual comments (1.07), and the GOTO statement (1.02). (See Appendix Figure 9.)